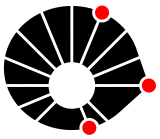

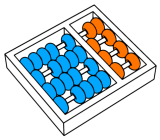

Avaliação do impacto da metodologia Computer Science Peer Instruction (CSPI) e da Taxonomia de Bloom na disciplina Introdutória CS1

Igor M. Omote *Ricardo E. Caceffo*

Relatório Técnico - IC-PFG-19-03
Projeto Final de Graduação
2019 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Avaliação do impacto da metodologia Computer Science Peer Instruction (CSPI) e da Taxonomia de Bloom na disciplina Introdutória CS1

Igor M. Omote* Ricardo E. Caceffo†

Resumo

Este trabalho tem como principal objetivo analisar os resultados do CSPI (*Computer Science Peer Instruction*), uma adaptação do *Peer Instruction* (PI) para disciplinas introdutórias de programação (CS1). O estudo, com duração de um semestre, foi realizado na disciplina MC102, ministrada de forma coordenada para cursos de *STEM* na Unicamp, abrangendo 8 turmas e 766 alunos. Utilizou-se como base de análise um conjunto de questões de múltipla-escolha (N = 10), denominadas de Atividades Conceituais (ACs) a respeito do tópico de ordenação. Os alunos foram divididos em dois grupos: grupo 1 (N = 127 alunos), participantes de uma turma onde o CSPI foi aplicado e; grupo 2 (N = 639), participantes das 7 turmas onde a aula tradicional foi mantida. Resultados indicam que o desempenho dos alunos nas ACs foi similar em ambos os grupos. Posteriormente, um conjunto de 15 questões de múltipla-escolha, baseadas na Taxonomia de Bloom, foi criado também para o tópico de ordenação, sendo aplicado de forma online para ambos os grupos. Resultados indicam que os alunos que tiveram aula suportadas com CSPI tiveram um desempenho 27% superior ao demais.

1 Introdução

Dentre as diversas formas de lecionar uma aula, uma das mais tradicionais é a *Lecture Learning*, na qual o aluno adquire conhecimento passivamente enquanto o professor palestra. Uma outra forma que vem crescendo no cenário educacional é o aprendizado ativo [1], no qual o aluno é o centro do aprendizado pois, além de assistir a uma aula, é incentivado a ler, discutir e resolver problemas.

Este trabalho teve como objetivo estudar a aplicação e resultados de uma metodologia de *Active Learning*, o o *Computer Science Peer Instruction* (CSPI) que será melhor detalhado na seção 2.2, aplicado na matéria de CS1 da Universidade Estadual de Campinas, chamada de MC102 - Algoritmos e Programação de Computadores [2].

Ademais, estudou-se formas de avaliar o aprendizado dos alunos, como a Taxonomia de Bloom [3] [4] e a identificação de *misconceptions* [5] para entender todos os impactos da aplicação da metodologia.

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP. Pesquisa desenvolvida com suporte financeiro parcial do PIBIC

†Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP. Processo número 2014/07502-4, Fundacao de Amparo a Pesquisado Estado de Sao Paulo (FAPESP)

2 Metodologia

A fim de facilitar o entendimento, esta seção está dividida da seguinte forma: a subseção **2.1** apresenta uma introdução histórica referente ao que será abordado; por sua vez, na subseção **2.2** é uma introdução ao conceito de Peer Instruction [1] com uso de *clickers* [6]; a subseção **2.3** discute uma análise do desempenho dos alunos do Professor Ricardo Caceffo que aplicou a metodologia de *Peer Instruction*; e, por fim, a subseção **2.4** apresentará como foi construída a proposta para melhorar a avaliação dos alunos dentro da taxonomia de Bloom e os resultados obtidos após a aplicação pela plataforma *moodle* [7]. O trabalho completo pode ser visto na Figura 1.

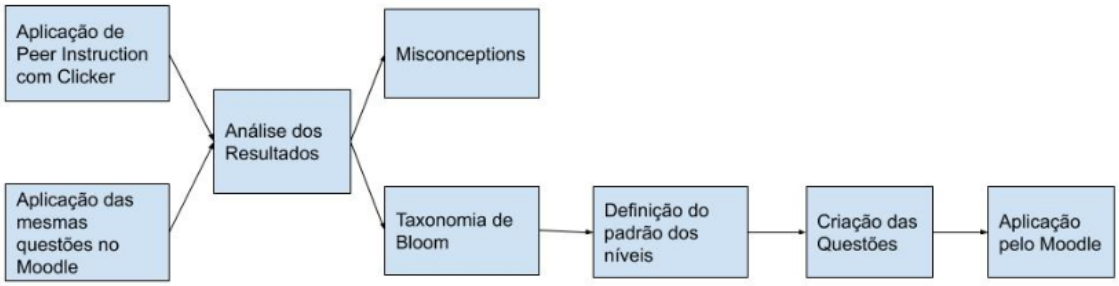

Figura 1: Fluxograma de trabalho seguido

2.1 Histórico da Taxonomia de Bloom

Conhecida inicialmente por Taxonomia dos objetivos educacionais, a taxonomia de Bloom foi proposta em 1956 por Benjamin S. Bloom em sua publicação "Taxonomy of educational objectives. The classification of educational goals" [3]. Trata-se de uma padronização em níveis de aprendizado em três domínios:

- Cognitivo - Objetivos atrelados a lembrar e reconhecer conhecimento e desenvolver habilidades intelectuais.
- Afetivo - Objetivos relacionados ao interesse, atitude, valores e desenvolver o apreço.
- Psicomotor - Objetivos relacionados a execução de tarefas que envolvam o sistema motor.

Nessa publicação, o objetivo principal de Bloom era facilitar a comunicação entre professores, ao padronizar e tornar as discussões acerca dos objetivos educacionais mais diretas e assertivas junto a uma escala que permite a troca de experiências e informações.

Em 2001, um dos alunos de Bloom juntamente com um dos pesquisadores originais, Lorin Anderson e David Krathwohl publicaram o livro "A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives" [4] em que revisaram a abordagem de Bloom com foco no domínio cognitivo, resumida na Tabela 1

A forma como Bloom (e posteriormente Anderson e Krathwohl [4]) dividiu os objetivos educacionais foi ampla e genérica, de forma que consiga abrangir qualquer área de conhecimento. Assim,

Tabela 1: Descrição de cada nível segundo a revisão de Anderson e Krathwohl

Nível Taxonomia	Descrição
Remember	Retrieve relevant knowlegd from long-term memory
Understand	Construct meaning from instructional messages, including oral, written, and graphic communication
Apply	Carry out or use a procedure in a given situation
Analyze	Break material into its constituent parts and determine how the parts relate to one another and to an overall structure or purpose
Evaluate	Make judgments based on criteria and standards
Create	Put elements together to form a coherent or function whole; reorganize elemnts into a new pattern or structure

é possível encontrar trabalhos nas mais diversas áreas, dentre elas, na Ciência da Computação. Alguns pesquisadores já tentaram utilizar a taxonomia de Bloom para CS1, que é a matéria introdutória aos primeiros conceitos de Ciência da Computação como pode ser visto em [8] e [9], entretanto encontraram algumas dificuldades como a dificuldade em elaborar questões para os níveis mais altos da taxonomia, visto em [8] e percebeu-se que pesquisadores com contextos diferentes classificam questões em níveis diferentes da taxonomia [9].

2.2 Peer Instruction suportado por Clicker

Peer Instruction (PI) é uma forma de aprendizado ativo introduzida e popularizada pelo Eric Mazur [1] em que o aluno responde a questões durante a aula. Inicialmente utilizada em conteúdos de física na Universidade de Harvard, mas popularizada em várias áreas da ciência, engenharia, matemática e tecnologia, o *Peer Instruction* facilita aluno e professor identificar falhas no aprendizado de forma mais rápida e iterativa. Dada uma questão, a aula só irá prosseguir caso **X**% dos alunos tenham acertado, caso contrário, é incentivado que os alunos discutam as respostas dadas e uma nova questão é apresentada até que o *threshold* de **X**% seja atingido.

Por esse motivo, faz-se necessário uma ferramenta para correção automática para que o docente não tenha de desperdiçar tempo corrigindo, mas lecionando. Assim, surgiu o uso de *clickers* para auxiliar na correção automática das questões apresentadas em aula. Como utilizado em [10]: uma questão é apresentada no slide durante a aula, os alunos respondem à questão pelo dispositivo e, assim que o tempo pré-determinado passar, o professor apresenta os resultados em sala, de acordo com o resultado, a aula prossegue ou não.

O professor Ricardo Caceffo aplicou tal metodologia no primeiro semestre de 2018 nas turmas 4,5,6 e 7 de MC102 (Algoritmos e Programação de Computadores), aula correspondente a CS1 na Universidade Estadual de Campinas. Os resultados serão apresentados na seção 3.

2.3 Análise do desempenho dos Alunos

Em um primeiro momento, foram analisadas duas bases de dados referentes às aulas 20 e 21, cujo tópico é **ordenação**: os dados dos *clickers*, como explicado na sub-seção 2.2 com uma média de dezessete (17) respostas por questão e dados da plataforma online *moodle*, em que as mesmas questões

da aula do professor Ricardo Caceffo foram apresentadas aos demais alunos do curso de MC102, com 473 respostas, sendo 90 dessas, alunos do professor Ricardo Caceffo.

Os dados dos *clickers* eram bem limitados, contendo apenas o número de pessoas que tentaram responder a questão e quantas pessoas assinalaram cada uma das possibilidades (A, B, C, D ou E), mas sem nenhuma identificação de quem foi o aluno ou quanto tempo levou para respondê-la. Dessa forma, apesar dos dados do *moodle* possuirem mais informação (como identificar qual alternativa cada um dos alunos escolheu, quantas vezes tentou responder ao questionário), não era possível fazer uma comparação direta.

Para contornar isso, analisou-se questão a questão, verificando seu correspondente nos dados do *clicker* e comparou-se a média de acertos em cada questão gerando um HeatMap de cada aula, um exemplo pode ser visto na tabela 6.

Na análise do *moodle* tomou-se a decisão de descartar todos os alunos com índice de acerto inferior a 20%, por ser o *threshold* da aleatoriedade em uma questão com cinco alternativas, tal refinamento não foi possível fazer com os dados dos *clickers*.

Além disso, uma vez que o *moodle* era aplicado a todas as turmas de MC102, separou-se os alunos do professor Ricardo Caceffo, já que estes potencialmente haviam tido contato com as questões em sala de aula (não é possível afirmar isso, uma vez que não houve controle de quais alunos utilizaram o *clicker*).

Em seguida, baseado no trabalho do Professor Ricardo Caceffo *et. al* com a identificação de falhas de aprendizado (*misconceptions*) para Python [5], foi possível perceber que faltavam questões tanto nos slides, quanto no *moodle* para identificar essas falhas nos alunos. Além disso, para complementar a análise do aprendizado dos alunos como um todo, fez-se uma tentativa de classificar a forma como a matéria de MC102 encaixa-se na Taxonomia de Bloom, partindo das questões previamentes apresentadas. Isso não foi possível pois as questões apresentadas não seguiam um padrão nem tinham relação direta com o proposto por Bloom em seu trabalho original.

2.4 Proposta para melhorar a avaliação dos alunos

Dada a situação apresentada anteriormente em 2.3, havia duas frentes de trabalho a seguir: criar questões que identificassem falhas de aprendizado continuamente (seja por *Peer Instruction*, ou pelo próprio *moodle*) para que o docente tenha um *feedback* contínuo do aprendizado em cada tópico e da evolução dos alunos. E elaborar questões que identifiquem como a matéria de MC102 é aplicada pelo desempenho médio dos alunos segundo a taxonomia de Bloom.

Este relatório técnico irá apresentar os resultados obtidos na segunda frente de trabalho. Uma vez que a primeira ainda está em elaboração até o presente momento que o relatório está sendo escrito.

Para tal, estudou-se trabalhos anteriores, extraindo duas principais ideias: criar uma tabela com cada um dos níveis da taxonomia [9] e definir um contexto base para alinhar a expectativa do que é esperado de uma aula padrão de ordenação para MC102. Adicionalmente, todas as questões criadas deveriam ser de correção automática para que, futuramente, todos professores tenham um *feedback* instantâneo.

Assim, criou-se *guidelines* para criar questões em cada nível da Taxonomia de Bloom especificadas na Tabela 8, com exceção do nível *Create*, em que os pesquisadores concluíram não haver forma de criar questões com correção automática, pois esse nível só é atingido quando se tem a capacidade de montar partes de conhecimento formando um novo todo.

Tabela 2: Número de alunos que responderam presencialmente, nas aulas com clickers, as questões referentes à Aula 20

#Questão	A	B	C	D	E	% acerto
L20-Q01	2	14	2	3	0	14.29%
L20-Q02	1	1	0	6	12	60.00%
L20-Q03	0	0	10	2	5	29.41%
L20-Q04	0	1	13	0	1	86.67%
L20-Q05	0	0	10	4	1	66.67%
L20-Q06	0	0	0	0	7	100.00%

Tabela 3: Número de alunos que responderam presencialmente, nas aulas com clickers, as questões referentes à Aula 21

#Questão	A	B	C	D	E	% acerto
L21-Q01	5	4	10	1	1	23,81%
L21-Q02	0	2	17	0	0	89,47%
L21-Q03	0	2	1	2	15	75,00%
L21-Q04	0	4	7	2	2	46,67%

Após isso, definiu-se o que é esperado de uma aula padrão de **ordenação** e fez-se da metodologia de *brainstorming* para elaborar as questões sobre o mesmo tópico, com auxílio da própria coordenadora do curso de MC102 do semestre em que este relatório é escrito.

Em sequência, houve a aplicação das questões para todos os alunos, com um total de 168 respostas, sendo de 130 alunos diferentes, mas somente 68 respostas concluídas (era possível responder parcialmente ao questionário). Para a análise do desempenho, considerou-se apenas os alunos que concluíram o questionário pela primeira vez.

3 Resultados

Nesta seção primeiro serão apresentados todos os resultados obtidos durante a execução do projeto, incluindo tabelas geradas referentes exclusivamente ao tópico de **ordenação**, em que houve maior profundidade.

Seguindo a ordem apresentada na seção 2, o primeiro resultado é referente às respostas dos alunos do professor Ricardo Caceffo durante suas aulas de MC102 com a utilização dos *clickers*.

A partir das tabelas 2 e 3 é clara a evolução dos alunos nas questõs L20-Q01 para L20-Q02, L20-Q03 para L20-Q04, L20-Q05 para L20-Q06. Nas tabelas A-1 e A-2 em anexo estão detalhados os enunciados e alternativas de cada questão.

Fato que não ocorre pela plataforma *moodle*, em que as mesmas questões foram dadas à todos alunos, tanto para turmas do professor Ricardo, quanto nas demais. Os resultados podem ser vistos nas tabelas 4 e 5.

A partir dos resultados das tabelas 2, 3, 4 e 5, foram feitas duas (2) novas tabelas comparando somente a porcentagem de acerto nas aulas 20 (Tabela 6) e 21 (Tabela 7 em um *HeatMap*, em que a

Tabela 4: Comparação entre alunos do Profº Ricardo Caceffo e demais alunos no *Moodle* Aula 20 20

	Turmas do Profº Ricardo	Demais Turmas	Todos alunos
Número de Alunos	90	383	473
Acertos L20-Q01	66 (73.33%)	239 (62.40%)	305 (64.48%)
Acertos L20-Q02	66 (73.33%)	261 (68.15%)	327 (69.13%)
Acertos L20-Q03	60 (66.67%)	260 (67.89%)	320 (67.65%)
Acertos L20-Q04	80 (88.89%)	326 (85.12%)	406 (85.84%)
Acertos L20-Q05	70 (77.78%)	303 (79.11%)	373 (78.86%)
Acertos L20-Q06	66 (73.33%)	281 (73.37%)	347 (73.36%)

Tabela 5: Comparação entre alunos do Profº Ricardo Caceffo e demais alunos no *Moodle* Aula 21 20

	Turmas do Profº Ricardo	Demais Turmas	Todos alunos
Número de Alunos	86	353	439
Acertos L21-Q01	53 (61.63%)	259 (73.37%)	312 (71.07%)
Acertos L21-Q02	79 (91.86%)	316 (89.52%)	395 (89.98%)
Acertos L21-Q03	71 (82.56%)	281 (79.60%)	352 (80.18%)
Acertos L21-Q04	67 (77.91%)	258 (73.09%)	325 (74.03%)

Tabela 6: HeatMap referente a Aula 20

	Clicker	Moodle		
	Turmas do Profº Ricardo	Turmas do Profº Ricardo	Demais Turmas	Todos alunos
L20-Q01	14.29%	73.33%	62.40%	64.48%
L20-Q02	60.00%	73.33%	68.15%	69.13%
L20-Q03	29.41%	66.67%	67.89%	67.65%
L20-Q04	86.67%	88.89%	85.12%	85.84%
L20-Q05	66.67%	77.78%	79.11%	78.86%
L20-Q06	100.00%	73.33%	73.37%	73.36%

Tabela 7: HeatMap referente a Aula 21

	Clicker	Moodle		
	Turmas do Profº Ricardo	Turmas do Profº Ricardo	Demais Turmas	Todos alunos
L21-Q01	23.81%	61.63%	73.37%	71.07%
L21-Q02	89.47%	91.86%	89.52%	89.98%
L21-Q03	75.00%	82.56%	79.60%	80.18%
L21-Q04	46.67%	77.91%	73.09%	74.03%

cor vermelha é referente a taxas de acerto inferiores a 50%, amarelo para maiores ou iguais a 50% e inferiores a 75% e verde para taxas maiores ou iguais a 75%. É evidente que a taxa de acerto pelos *clikers* é inferior ao da plataforma *moodle*. Além disso, quando se compara somente na plataforma *moodle*, as turmas do professor Ricardo Caceffo tiveram maior taxa de acerto do que as demais. Mais detalhes serão discutidos na seção 4.

Por fim, houve a tentativa de classificar as turmas de MC102 nos níveis da taxonomia de Bloom, mas como as questões não foram planejadas para tal, a maioria das questões pertenciam aos níveis *Understand* e *Apply*, impedindo a classificação em níveis inferiores (*Remember*) ou superiores (*Analyze* e *Evaluate*).

Assim, criou-se um conjunto de questões referentes ao tópico de **ordenação**, sendo quatro referente ao nível *Remember*, quatro no nível *Understand*, duas no nível *Apply*, quatro no nível *Analyze* e uma no nível *Evaluate* (Tabela A-3) que conseguissem classificar dentro de toda taxonomia de Bloom. Para tal, desenvolvou-se uma padronização (Tabela 8) para que os problemas apontados em [9] fossem corrigidos.

Posteriormente, as questões foram aplicadas nos alunos da turma de MC102 do primeiro semestre de 2019 pela plataforma *moodle*, utilizando a mesma metodologia citada em 2.3. O resultado segue na tabela 9

4 Discussão

Em relação à Aula 20 do professor Ricardo Caceffo, é possível perceber pela Tabela A-1 que as questões L20-Q01 e L20-Q02, L20-Q03 e L20-Q04, L20-Q05 e L20-Q06 são pares que tratam do mesmo assunto dentro do tópico **ordenação**. Sendo assim, o aumento da porcentagem de acertos

Tabela 8: Padronização para a criação de questões na taxonomia de Bloom

Remember	Understand	Apply	Analyze	Evaluate
Conceitos decorado pelos alunos	Reconhecimento de código simples	Executar o algoritmo passo a passo	Comparação de um conjunto de algoritmos em relação à sua complexidade (número de trocas ou comparações)	Tomar uma decisão e justificá-la
Está associado ao que foi ensinado em sala de aula	Texto na língua em que a aula foi lecionada; descrevendo o algoritmo			Questões abertas ou que exigem um raciocínio maior do aluno (e.g. drag and drop)
Devem ser no idioma em que a aula foi lecionada; em alto nível	Vídeos de simulação de algoritmos			

Tabela 9: Resultado das questões relacionadas a taxonomia de Bloom

	Turma Prof Ricardo Caceffo	Demais	Total
% Acerto Remember	75.00%	58.20%	59.93%
% Acerto Understand	78.57%	52.46%	55.15%
% Acerto Apply	28.57%	22.13%	22.79%
% Acerto Analyze	53.57%	46.72%	47.43%
% Acerto Evaluate	71.43%	62.30%	63.24%

da primeira questão para a segunda é um indício da evolução dos alunos após a discussão entre eles como previsto no *Peer Instruction*.

Já na aula 21, representada na Tabela A-2, percebe-se uma queda na taxa de acertos da L21-Q03 para a L21-Q04, isso ocorre pois a L21-Q04 encaixa-se no nível *Analyze* da taxonomia de Bloom, e nenhuma questão apresentada anteriormente alcançou tal nível, assim, era esperado uma queda no índice de acerto. Entretanto, uma vez que não houve uma segunda questão nesse nível, não é possível afirmar se o *Peer Instruction* novamente irá melhorar a performance dos alunos.

Analisando os resultados da plataforma *moodle* nas Tabelas 4 e 5, a diferença de acertos entre os alunos do Professor Ricardo Caceffo com os demais é pequena, sendo que as turmas deste atingiram uma média de 76.69%, enquanto as demais turmas atingiram 75.04%. Entretanto, unindo as tabelas ao gerar os *heatmaps* das Tabelas 6 e 7, observa-se aumento significativo nos resultados das turmas do professor em questão.

Criou-se duas (2) hipóteses para explicar o ocorrido: os alunos podem ter decorado as respostas das questões, mas é importante ressaltar que há uma diferença considerável entre o número de alunos que responderam em sala (aproximadamente 17 alunos) e que responderam no *moodle* (em torno de 88 alunos) ou o *Peer Instruction* fixou melhor o conteúdo nos alunos.

A partir da hipótese anterior e das questões criadas para mapear os níveis da taxonomia de Bloom, chegou-se na Tabela 9, nela é possível perceber um desempenho consistentemente melhor nas turmas do Professor Ricardo Caceffo em todos os níveis. E, poderando a diferença de acertos pelo número de questões em cada nível, tem-se um aumento de 27.02% nas turmas que foram suportadas pelo *Peer Instruction*. Vale ressaltar que nenhuma turma havia tido contato com as questões elaboradas, logo, o *Peer Instruction* melhorou o desempenho a longo prazo dos alunos.

Uma detalhe que chamou a atenção dos pesquisadores é o fato dos alunos terem desempenho melhor no nível *Analyze* e *Evaluate* do que no *Apply*, acredita-se que faltou deixar os códigos dos algoritmos de ordenação explícitos (como feito no nível de *Analyze*, vide Tabela A-3) para que os alunos os executassem, ou seja, criou-se mais uma camada de dificuldade que não existiu nas demais questões.

5 Conclusão

Como principal conclusão tem-se que a metodologia de *Peer Instruction* apresenta melhorias no aprendizado em sala de aula, resultando em melhores resultados na Taxonomia de Bloom, principalmente nos níveis de *Remember* e *Understand*, o esperado era que o desempenho dos alunos fosse sendo decrementado conforme as questões evoluíam na Taxonomia de Bloom. Entretando, verificou-se que a menor taxa de acerto foi no nível *Apply*, ao passo que houve aumento nos acertos nos níveis *Analyze* e *Evaluate*

Além disso, uma vez definida, a padronização do que é esperado em cada nível da taxonomia de Bloom para CS1 facilita a criação de questões e acaba com ambiguidade de definições. A forma como Anderson e Krathwohl [4] revisaram-na em 2001, foi genérica; cada especialista que deseja usá-la deve refinar para sua área de conhecimento e, a partir disso, criar o questionário.

6 Trabalhos Futuros

Para trabalhos futuros, tem-se a elaboração de questões para identificação das falhas de aprendizado (*misconceptions*), como mencionado na seção 2. Dessa forma, os professores terão um *feedback* completo da evolução dos alunos a cada aula e no curso como um todo.

Ademais, haverá um trabalho considerável para expandir as questões para todos tópicos de MC102 e entender o resultado obtido nos níveis de *Apply*, *Analyze* e *Evaluate* no tópico de **ordenação**, uma vez que era esperado que os alunos que tivessem bom desempenho nos dois últimos níveis citados, minimamente deveriam ter tido um bom desempenho no *Apply*.

Com os dois passos anteriores concluídos, será necessário automatizar a criação de relatórios com um resumo das falhas no aprendizado dos alunos e do nível de aprendizado que se encontram.

7 Agradecimentos

A elaboração do presente relatório teve diversas influências, por isso, agradeço:

Ao PIBIC (Programas de Iniciação Científica e Tecnológica) por financiar a pesquisa que gerou o relatório e ao processo número 2014/07052-4, Fundação de Amparo à Pesquisado Estado de São Paulo (FAPESP)

Ao professor doutor Ricardo Caceffo que auxiliou em toda a pesquisa e forneceu todos insumos necessários para tal e à professora Islene Calciolari Garcia pela confiança ao permitir aplicarmos nosso questionário com os alunos de MC102.

À Unicamp e todos docentes que tive durante minha graduação, que com suas particularidades me mostraram diversas formas de lecionar e aprender, e como cada uma delas é importante.

À minha família que deu todas as bases da minha educação, suporte nas maiores dificuldades e inspiração para superá-las.

E aos amigos que fiz durante essa trajetória que me mostraram a importância do trabalho em equipe e como é possível atingir resultados excepcionais quando se confia nos outros, todos foram essenciais na minha vida universitária.

Referências

[1] C. H. Crouch and E. Mazur, "Peer instruction: Ten years of experience and results," *American Journal of Physics*, vol. 69, no. 9, pp. 970–977, 2001. [Online]. Available: https://doi.org/10.1119/1.1374249

[2] Unicamp. Link para acessar página da ementa de mc102. [Online]. Available: https://www.ic.unicamp.br/ mc102/pdd.html

[3] B. S. Bloom, M. B. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl, *Taxonomy of educational objectives. The classification of educational goals. Handbook 1: Cognitive domain*. New York: Longmans Green, 1956.

[4] L. W. Anderson, D. R. Krathwohl *et al.*, "A revision of bloom's taxonomy of educational objectives," *A Taxonomy for Learning, Teaching and Assessing. Longman, New York*, 2001.

[5] G. Gama, R. Caceffo, R. Souza, R. Bennati, T. Aparecida, I. Garcia, and R. Azevedo, "An antipattern documentation about misconceptions related to an introductory programming course in python," Institute of Computing, University of Campinas, Tech. Rep. IC-18-19, November 2018.

[6] iClicker. Link para acessar página institucional do iclicker. [Online]. Available: https://www.iclicker.com/

[7] M. HQ. Link para acessar página institucional do moodle. [Online]. Available: https://docs.moodle.org/37/en/About_Moodle

[8] D. Oliver, T. Dobele, M. Greber, and T. Roberts, "This course has a bloom rating of 3.9," in *Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30*. Australian Computer Society, Inc., 2004, pp. 227–231.

[9] R. Gluga, J. Kay, R. Lister, S. Kleitman, and T. Lever, "Coming to terms with bloom: an online tutorial for teachers of programming fundamentals," in *Proceedings of the Fourteenth Australasian Computing Education Conference-Volume 123*. Australian Computer Society, Inc., 2012, pp. 147–156.

[10] R. Caceffo, G. Gama, and R. Azevedo, "Exploring active learning approaches to computer science classes," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: ACM, 2018, pp. 922–927. [Online]. Available: http://doi.acm.org/10.1145/3159450.3159585

Anexos

Anexo A

Neste anexo são apresentados as seguintes tabelas: a Tabela A-1 são as questões e alternativas apresentadas aos alunos na Aula 20; a Tabela A-2 são as questões e alternativas apresentadas aos alunos na Aula 21; por fim, a Tabela A-3 apresenta as questões e alternativas elaboradas pela equipe em cada nível da Taxonomia de Bloom.

Tabela A-1: Questões e alternativas da Aula 20

L20-Q01	O que o programa irá exibir?
	`#Dada uma posicao de inicio,`
	`#retorna o INDICE do menor numero`
	`def indiceMenor(lista, inicio):`
	`  indice_menor = inicio`
	`  for i in range(inicio, len(lista)):`
	`  if lista[i] > lista[indice_menor]:`
	`  indice_menor = i`
	`  return indice_menor`
	`lista = [-123, 2, 9, 7, 5, -18, -10, 4]`
	`menor = indiceMenor(lista, 0)`
	`print("menor__=_", menor)`
		a) Não irá compilar.
		b) menor = -123
		c) menor = 0
		d) menor = 2
		e) menor = 9

L20-Q02	O que o programa irá exibir?
	`#Dada uma posicao de inicio,`
	`#retorna o INDICE do menor numero`
	`def indiceMenor(lista, inicio):`
	`  indice_menor = inicio`
	`  for i in range(inicio, len(lista)):`
	`  if lista[i] < lista[indice_menor]:`
	`  indice_menor = i`
	`  return indice_menor`
	`lista = [-123, 2, 9, 7, 5, -18, -10, 4]`
	`menor = indiceMenor(lista, 1)`
	`print("menor__=_", menor) :`
		a) Não irá compilar.
		b) menor = -18
		c) menor = 123
		d) menor = 0
		e) menor = 5

<table>
<tr>
<td>L20-Q03</td>
<td>O que o programa irá exibir?

```
#Dada uma posicao de inicio, retorna o
#INDICE do menor numero
def indiceMenor(lista, inicio):
    indice_menor = inicio
    for i in range(inicio, len(lista)):
        if lista[i] < lista[indice_menor]:
            indice_menor = i
return indice_menor
lista = [-3, 0, -19]
for i in range (len(lista)-1):
    min = indiceMenor(lista,i)
    if (lista[min] < lista[i]):
        #Troca lista[min] com lista[i]
        lista[min] = lista[i]
        lista[i] = lista[min]
print(lista)
```

</td>
<td>a) Não irá compilar.
b) menor = [0,3,-19]
c) menor = [-19,-3,0]
d) menor = [0,-3,19]
e) menor = [-3,0,0]</td>
</tr>
<tr>
<td>L20-Q04</td>
<td>O que o programa irá exibir?

```
#Dada uma posicao de inicio, retorna o
#INDICE do menor numero
def indiceMenor(lista, inicio):
    indice_menor = inicio
    for i in range(inicio, len(lista)):
        if lista[i] < lista[indice_menor]:
            indice_menor = i
    return indice_menor
    lista = [-3, 0, -19]
for i in range (len(lista)-1):
    min = indiceMenor(lista,i)
    if (lista[min] < lista[i]):
        #Troca lista[min] com lista[i]
        aux = lista[min]
        lista[min] = lista[i]
        lista[i] = aux
print(lista)
```

</td>
<td>a) Não irá compilar.
b) menor = [0,3,-19]
c) menor = [-19,-3,0]
d) menor = [0,-3,19]
e) menor = [-3,0,0]</td>
</tr>
</table>

L20-Q05	O que o programa irá exibir?
	`#Dada uma posicao de inicio, retorna o`
	`#INDICE do menor numero`
	`def indiceMenor(lista, inicio):`
	`  indice_menor = inicio`
	`  for i in range(inicio, len(lista)):`
	`  if lista[i] < lista[indice_menor]:`
	`  indice_menor = i`
	`  return indice_menor`
	`lista = [-3, 5, 2, 1, 5]`
	`trocas = 0`
	`for i in range (len(lista)-1):`
	`  min = indiceMenor(lista,i)`
	`  if (lista[min] < lista[i]):`
	`  #Troca lista[min] com lista[i]`
	`  trocas = trocas + 1`
	`  aux = lista[min]`
	`  lista[min] = lista[i]`
	`  lista[i] = aux`
	`print("trocas ", trocas)`
		a) Não irá compilar.
		b) trocas 0
		c) trocas 1
		d) trocas 2
		e) trocas 4

L20-Q06	O que o programa irá exibir?
	`#Dada uma posicao de inicio, retorna o INDICE`
	`#do menor numero`
	`def indiceMenor(lista, inicio):`
	`  indice_menor = inicio`
	`  for i in range(inicio, len(lista)):`
	`  if lista[i] <= lista[indice_menor]:`
	`  indice_menor = i`
	`  return indice_menor`
	`lista = [-3, 5, 2, 1, 5]`
	`trocas = 0`
	`for i in range (len(lista)-1):`
	`  min = indiceMenor(lista,i)`
	`  if (lista[min] < lista[i]):`
	`  #Troca lista[min] com lista[i]`
	`  trocas = trocas + 1`
	`  aux = lista[min]`
	`  lista[min] = lista[i]`
	`  lista[i] = aux`
	`print("trocas ", trocas)`
		a) Não irá compilar.
		b) trocas 0
		c) trocas 1
		d) trocas 2
		e) trocas 4

Tabela A-2: Questões e alternativas da Aula 21

L21-Q01	O programa abaixo implementa o algoritmo do INSERTION SORT. Quais linhas devem OBRIGATORIAMENTE serem removidas para que o programa funcione como esperado? 1. lista = [7, 8, 5, 2, 4, 6, 3] 2. i = 1 3. **for** i **in** **range**(1, **len**(lista)): 4. elem = lista[i] 5. local = i 6. **while** (local != 0 **and** lista[local−1] > elem): 7. lista[local] = lista[local−1] 8. local = local − 1 9. lista[local] = elem 10. i = i + 1 11. **print**("Lista ordenada = ", lista)	a) Nenhuma linha precisa (embora eventualmente possa) ser removida. Se o código for executado exatamente como está o programa irá funcionar como esperado. b) Linha 2 c) Linha 10 d) Linha 2 e Linha 10 e) Linha 8
L21-Q02	O seguinte programa supostamente implementa o algoritmo do INSERTION SORT. O que será exibido? lista = [2, −1, 0, 2] **for** i **in** **range**(1, **len**(lista)): elem = lista[i] local = i **while** (local != 0 **and** lista[local−1] > elem): lista[local] = lista[local−1] local = local − 1 lista[local] = elem **print**(lista)	a) Não irá compilar b) [2, -1, 0, 2] c) [-1, 0, 2, 2] d) [-1, 0, 2] e) [0,1, 2, 2]

L21-Q03	O seguinte programa implementa o algoritmo do INSERTION SORT. O que será exibido?
	`lista = [4, 1, 2]`
	`for i in range(1, len(lista)):`
	`  elem = lista[i]  local = i`
	`  while (local != 0 and lista[local-1] > elem):`
	`  lista[local] = lista[local-1]`
	`  local = local - 1`
	`  lista[local] = elem`
	`  print(lista)`
		a) Não irá compilar
		b) [4, 1, 2]
		c) [1, 4, 2]
		d) [1, 2, 4]
		e) [4, 2, 1]

L21-Q04	O seguinte programa implementa corretamente o algoritmo do INSERTION SORT, ordenando de forma crescente uma lista de inteiros. A variável *deslocamento* conta quantas vezes um determinado número foi deslocado para uma posição subsequente (à sua direita). O que será exibido quando o programa for executado?
	`lista = [7, 8, 5, 2, 4, 6, 3]`
	`deslocamento = 0`
	`for i in range(1, len(lista)):`
	`  elem = lista[i]`
	`  local = i`
	`  while (local != 0 and lista[local-1] > elem):`
	`  deslocamento += 1`
	`  lista[local] = lista[local-1]`
	`  local = local - 1`
	`  lista[local] = elem print(deslocamento)`
		a) 10
		b) 13
		c) 15
		d) 17
		e) 20

Tabela A-3: Questões elaboradas para mapear níveis na Taxonomia de Bloom

Nível	Questão	Alternativas
Remember	Dada uma lista *lista* = [2, 3, 4, 1, 5], ao passarmos *lista* como parâmetro de uma função foo: x = *foo*(lista) O valor de x é 5, e *lista* não foi alterada. A função foo **não** pode ser um algoritmo de:	a) Identificar o maior elemento b) Ordenar uma lista c) Retornar o tamanho da lista d) Buscar a posição do maior elemento e) Não sei
Remember	Assinale as caixas com os algoritmos de ordenação, ou deixe todas vazias, se for o caso. Escolha uma ou mais:	☐ Identificar o maior elemento ☐ Ordenar uma lista ☐ Retornar o tamanho da lista ☐ Buscar a posição do maior elemento

Remember	Qual o efeito dos algoritmos *Selection Sort, Bubble Sort, Insertion Sort* vistos em sala de aula?	a) Identificar o maior elemento b) Ordenar uma lista c) Retornar o tamanho da lista d) Buscar a posição do maior elemento e) Não sei
Remember	"Suponho que a lista está ordenada até certo ponto, pegue o próximo elemento e insira na posição adequada” Assinale a alternativa do algoritmo que o trecho anterior faz parte:	a) Insertion Sort b) Selection Sort c) Busca Binária d) Bubble Sort e) Não sei

Understand	A seguinte descricao se refere a qual algoritmo de ordenação? "Transfira o menor valor da lista para a primeira posição, depois o segundo menor valor para a segunda posição, e assim, sucessivamente com os elementos restantes, por fim, tem-se a lista ordenada.”	a) Selection Sort b) Insertion Sort c) Bubble Sort d) Quick Sort e) Não sei
Understand	A seguinte descricao se refere a qual algoritmo de ordenação? "Percorra a lista de itens do início ao fim, comparando os valores dos elementos dois a dois, e trocando-os de lugar se necessário. Percorra a lista até que nenhum elemento tenha sido trocado na passagem anterior”	a) Selection Sort b) Insertion Sort c) Bubble Sort d) Quick Sort e) Não sei
Understand	A seguinte descricao se refere a qual algoritmo de ordenação? "Por definição, a primeira posição da lista já está ordenada (uma vez que é uma sub-lista única), para os elementos seguintes, coloque-os nas posições adequadas na sub-lista previamente ordenada”	a) Selection Sort b) Insertion Sort c) Bubble Sort d) Quick Sort e) Não sei

Understand	Sendo **a** = 5, **b** = 3
	Em Python, a operação:
	a, b = b, a
	Irá trocar os valores de **a** e **b**, resultando em **a** = 3, **b** = 5
	`1. def algoritmo(lista):`
	`2.  for i in range(len(lista) - 1):`
	`3.  minimo = i`
	`4.  for j in range (i, len(lista)):`
	`5.  if lista[min] > lista[j]:`
	`6.  minimo = j`
	`7.  lista[minimo], lista[i] = lista[i], lista[minimo]`
	O algoritmo acima é um exemplo de:
		a) Selection Sort
		b) Insertion Sort
		c) Bubble Sort
		d) Quick Sort
		e) Não sei

Apply	Dada uma lista *lista* = [3,1,5,4,2] A sequência de iterações a seguir são de qual Algoritmo de ordenação? *lista* = [3, 1, 5, 4, 2] *lista* = [1, 3, 5, 4, 2] *lista* = [1, 3, 5, 4, 2] *lista* = [1, 3, 4, 5, 2] *lista* = [1, 2, 3, 4, 5]	a) Selection Sort b) Insertion Sort c) Bubble Sort d) Quick Sort e) Não sei

Apply	Considere o algoritmo abaixo para a próxima questão:
	`1. for i in range(len(lista)):`
	`2.  min_idx = i`
	`3.  for j in range(i+1, len(lista)):`
	`4.  if lista[min_idx] > A[j]:`
	`5.  min_idx = j`
	`6.  lista[i], lista[min_idx] = lista[min_idx], lista[i]`
	Preencha as caixas com o valor que a variável min_idx assume ao final das duas primeiras execuções do algoritmo de ordenação acima para a lista: lista = [3, 4, 1, 2]
	min_idx = ▭
	min_idx = ▭
		min_idx = ▭ (Opções: 1, 2, 3, 4)
		min_idx = ▭ (Opções: 1, 2, 3, 4)

<table>
<tr>
<td>Analyze</td>
<td>Considere os algoritmos baixo:

```
1.    def selection_sort(lista):
2.        for i in range(len(lista)):
3.            min_idx = i
4.            for j in range(i+1, len(lista)):
5.                    if lista[min_idx] > A[j]:
6.                        min_idx = j
7.            A[i], A[min_idx] = A[min_idx], A
    [i]
```

```
1.    def insertion_sort(lista):
2.        for i in range(1, len(lista)):
3.            chave = lista[i]
4.            j = i - 1
5.            while(j >= 0 and key < lista[j])
    :
6.                    lista[j+1] = lista[j]
7.                    j = j - 1
8.            lista[j+1] = chave
```

```
1.    def bubble_sort(lista):
2.        tam = len(lista)
3.        for i in range(tam):
4.            for j in range(0, n - i - 1):
5.                if (lista[j] > lista[j + 1])
    :
6.                    lista[j], lista[j + 1] =
    lista[j], lista[j + 1]
```

E, considere como uma única operação de atribuição qualquer uma das operações abaixo:

```
1. a = b
2. a, b = b, a
```

Sendo assim, dado a lista **lista** = [8, 2, 3, 4, 5, 6, 7, 1], qual dos algoritmos irá executar o **menor** número de atribuições e retornar a lista ordenada? Escolha uma:</td>
<td>a) Selection Sort
b) Insertion Sort
c) Bubble Sort
d) Não sei</td>
</tr>
</table>

Analyze	Considere os algoritmos baixo:
	`1. def selection_sort(lista):`
	`2. for i in range(len(lista)):`
	`3. min_idx = i`
	`4. for j in range(i+1, len(lista)):`
	`5. if lista[min_idx] > A[j]:`
	`6. min_idx = j`
	`7. A[i], A[min_idx] = A[min_idx], A[i]`
	
	`1. def insertion_sort(lista):`
	`2. for i in range(1, len(lista)):`
	`3. chave = lista[i]`
	`4. j = i - 1`
	`5. while(j >= 0 and key < lista[j]):`
	`6. lista[j+1] = lista[j]`
	`7. j = j - 1`
	`8. lista[j+1] = chave`
	
	`1. def bubble_sort(lista):`
	`2. tam = len(lista)`
	`3. for i in range(tam):`
	`4. for j in range(0, n - i - 1):`
	`5. if (lista[j] > lista[j + 1]):`
	`6. lista[j], lista[j + 1] = lista[j], lista[j + 1]`
	
	E, considere como uma única operação de atribuição qualquer uma das operações abaixo:
	`1. a = b`
	`2. a, b = b, a`
	
	Sendo assim, dado a lista **lista** = [8, 7, 6, 5, 4, 3, 2, 1], qual dos algoritmos irá executar o **maior** número de atribuições e retornar a lista ordenada? Escolha uma:
		a) Selection Sort
		b) Insertion Sort
		c) Bubble Sort
		d) Não sei

Analyze	Dado a lista completamente desordenada: $list = [4, 3, 2, 1]$ Qual dos algoritmos abaixo irá executar o **menor** número de trocas entre posições Escolha uma:	a) Selection Sort b) Insertion Sort c) Bubble Sort d) Não sei
Analyze	Dado a lista completamente desordenada: $list = [4, 3, 2, 1]$ Qual dos algoritmos abaixo irá executar o **maior** número de trocas entre posições Escolha uma:	a) Selection Sort b) Insertion Sort c) Bubble Sort d) Não sei
Evaluate	Baseado nas respostas anteriores, para a lista quase ordenada: **lista = [1, 6, 3, 4, 5, 2]**. Preencha os campos a seguir com qual algoritmo de ordenação iria utilizar e o porquê. Irei utilizar o algoritmo ▭ devido ao ▭ número de ▭ necessárias.	Irei utilizar o algoritmo ▭ (Opções: Selection Sort, Insertion Sort, Bubble Sort) devido ao ▭ (Opções: menor, maior) número de ▭ (Opções: trocas, comparações) necessárias.