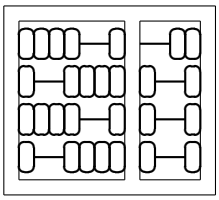

MC202 – Laboratório 07
Comparando algoritmos de ordenação
Instituto de Computação — UNICAMP
Prof.: Anderson Rocha
anderson.rocha@ic.unicamp.br

Objetivos

Implementar quatro algoritmos de ordenação (*insertion sort*, *heapsort*, *quicksort* e, usando esses três, o *introsort*) e mostrar quantas comparações e trocas cada um faz ao ordenar vetores.

Especificação

1. A primeira entrada para seu programa será um número inteiro positivo que indicará o número de elementos a serem lidos e ordenados em **ordem crescente**. Em seguida vem um `\n` e uma lista de inteiros separados por um espaço.

2. Todos os algoritmos devem receber a mesma sequência para ser ordenada. A maneira mais fácil de garantir isso é fazer uma cópia da sequência antes de chamar cada função de ordenação, assim cada algorítimo pode modificar apenas a sua cópia. Todos os algoritmos devem ser executados *in-place*.

3. Ao final da execução seu programa deve imprimir o número de comparações e trocas realizadas por cada algorítimo ao ordenar o vetor. O formato será: `printf("%s\n comparacoes: %d\n trocas: %d\n", nome do algorítimo, comparações, trocas)`.

4. O *insertion sort* deve manter a invariante de loop de que todos os números a esquerda do número atual devem estar ordenados. O número atual deve ser levado até a sua posição final por meio de uma série de trocas, indo do maior para o maior (direita para esquerda), até que o elemento inserido seja maior do que o a sua esquerda.

5. O *heapsort* deve manter um *heap* de máximo, para poder a cada iteração pegar o maior valor do topo do *heap* e substituí-lo pelo último elemento do *heap* (na ordem do vetor). As trocas e comparações na construção e manutenção do *heap* devem ser contadas para o total.

6. O *quicksort* deve pegar como pivô o primeiro elemento do vetor/subvetor a ser ordenado, e movê-lo para a posição correta no final de cada operação de partição. Ordene recursivamente as duas metades via *quicksort* até o caso base trivial de um só elemento no subvetor.

7. O *introsort* é uma combinação dos três algoritmos acima.

   (a) Inicialmente, usa o método de partição e recursão do *quicksort* pois é, em geral, o método mais rápido para vetores médios e grandes.

   (b) Quando sobrar quatro ou menos elementos em um ramo da recursão, chamar o *insertion sort* para finalizar este pedaço da ordenação. Em implementações reais esse número seria entre 15 e 50.

   (c) Se a profundidade da recursão chegar a três e não for o caso ordenar pelo *insertion sort*, deve-se ordenar esse subvetor usando o *heapsort*. Isso garante um pior caso *O(n lg n)* para o *introsort*. Ao iniciar, a profundidade de recursão do *introsort* é zero. A profundidade de um *introsort* chamado recursivamente por um outro *introsort* é um a mais do que a do *introsort* que o chamou. Na realidade, o momento ideal de chamar o *heapsort* seria dependente do logaritmo do número total de elementos.

   Ou seja, a última etapa de ordenação será feita apenas pelo *insertion sort* ou *heapsort*, nunca pelo método do *quicksort*, pois esse tem um custo fixo maior.

Exemplo 1

Entrada:

```
5
2 4 3 1 5
```

Saída:

```
Insertion Sort
 comparacoes: 7
 trocas: 4
Heapsort
 comparacoes: 12
 trocas: 10
Quicksort
 comparacoes: 13
 trocas: 4
Introsort
 comparacoes: 8
 trocas: 2
```

Exemplo 2

Entrada:

```
10
3 2 7 1 6 9 9 4 8 0
```

Saída:

```
Insertion Sort
 comparacoes: 26
 trocas: 20
Heapsort
 comparacoes: 38
 trocas: 27
Quicksort
 comparacoes: 32
 trocas: 9
Introsort
 comparacoes: 27
 trocas: 11
```

Observações

1. Os laboratórios devem ser compiláveis usando o GCC 4, com a seguinte linha de execução: `gcc -lm -std=c99 -Wall -pedantic -Werror -o main arquivo.c`

2. É obrigatório liberar toda a memória alocada. Os programas que não liberarem a memória alocada serão considerados errados, independentemente da saída correta nos testes.

3. Códigos ilegíveis serão considerados errados. A legibilidade é obtida com identação correta e coerente, bons nomes de variáveis e funções, bem como boa subdivisão do código em funções auxiliares.