MC102 – Aula 27
Recursão III - MergeSort

Eduardo C. Xavier

Instituto de Computação – Unicamp

28 de Junho de 2017

Introdução

- Problema:
  - Temos um vetor **v** de inteiros de tamanho **n**.
  - Devemos deixar **v** ordenado crescentemente.
- Veremos um algoritmo baseado na técnica **dividir-e-conquistar** que usa recursão.

Introdução

- Problema:
  - Temos um vetor **v** de inteiros de tamanho **n**.
  - Devemos deixar **v** ordenado crescentemente.
- Veremos um algoritmo baseado na técnica **dividir-e-conquistar** que usa recursão.

Dividir e Conquistar

- Temos que resolver um problema P de tamanho n.
- Mostramos como resolver casos básicos, como quando $n = 1$.
- Para $n > 1$ fazemos:
  - **Dividir:** Quebramos P em sub-problemas menores.
  - Resolvemos os sub-problemas de forma recursiva (Hip. Ind.).
  - **Conquistar:** Unimos as soluções dos sub-problemas para obter solução do problema maior P.

Dividir e Conquistar

- Temos que resolver um problema P de tamanho n.
- Mostramos como resolver casos básicos, como quando $n = 1$.
- Para $n > 1$ fazemos:
  - **Dividir:** Quebramos P em sub-problemas menores.
  - Resolvemos os sub-problemas de forma recursiva (Hip. Ind.).
  - **Conquistar:** Unimos as soluções dos sub-problemas para obter solução do problema maior P.

Dividir e Conquistar

- Temos que resolver um problema P de tamanho n.
- Mostramos como resolver casos básicos, como quando $n = 1$.
- Para $n > 1$ fazemos:
  - **Dividir:** Quebramos P em sub-problemas menores.
  - Resolvemos os sub-problemas de forma recursiva (Hip. Ind.).
  - **Conquistar:** Unimos as soluções dos sub-problemas para obter solução do problema maior P.

Dividir e Conquistar

- Temos que resolver um problema P de tamanho n.
- Mostramos como resolver casos básicos, como quando $n = 1$.
- Para $n > 1$ fazemos:
  - **Dividir:** Quebramos P em sub-problemas menores.
  - Resolvemos os sub-problemas de forma recursiva (Hip. Ind.).
  - **Conquistar:** Unimos as soluções dos sub-problemas para obter solução do problema maior P.

Merge-Sort: Ordenação por intercalação

- O Merge-Sort é um algoritmo baseado na técnica dividir-e-conquistar.
- Neste caso temos que ordenar um vetor de tamanho n.
- Se $n = 1$ então o problema está resolvido pois o vetor está ordenado.
- Para $n > 1$ temos:
  - **Dividir:** Dividimos o vetor de tamanho n em dois sub-vetores de tamanho aproximadamente iguais (um de tamanho $\lceil n/2 \rceil$ e outro de tamanho $\lfloor n/2 \rfloor$).
  - Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores.
  - **Conquistar:** Com os dois sub-vetores ordenados, construímos um vetor de tamanho n ordenado.

Merge-Sort: Ordenação por intercalação

- O Merge-Sort é um algoritmo baseado na técnica dividir-e-conquistar.
- Neste caso temos que ordenar um vetor de tamanho n.
- Se $n = 1$ então o problema está resolvido pois o vetor está ordenado.
- Para $n > 1$ temos:
  - **Dividir:** Dividimos o vetor de tamanho n em dois sub-vetores de tamanho aproximadamente iguais (um de tamanho $\lceil n/2 \rceil$ e outro de tamanho $\lfloor n/2 \rfloor$).
  - Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores.
  - **Conquistar:** Com os dois sub-vetores ordenados, construímos um vetor de tamanho n ordenado.

Merge-Sort: Ordenação por intercalação

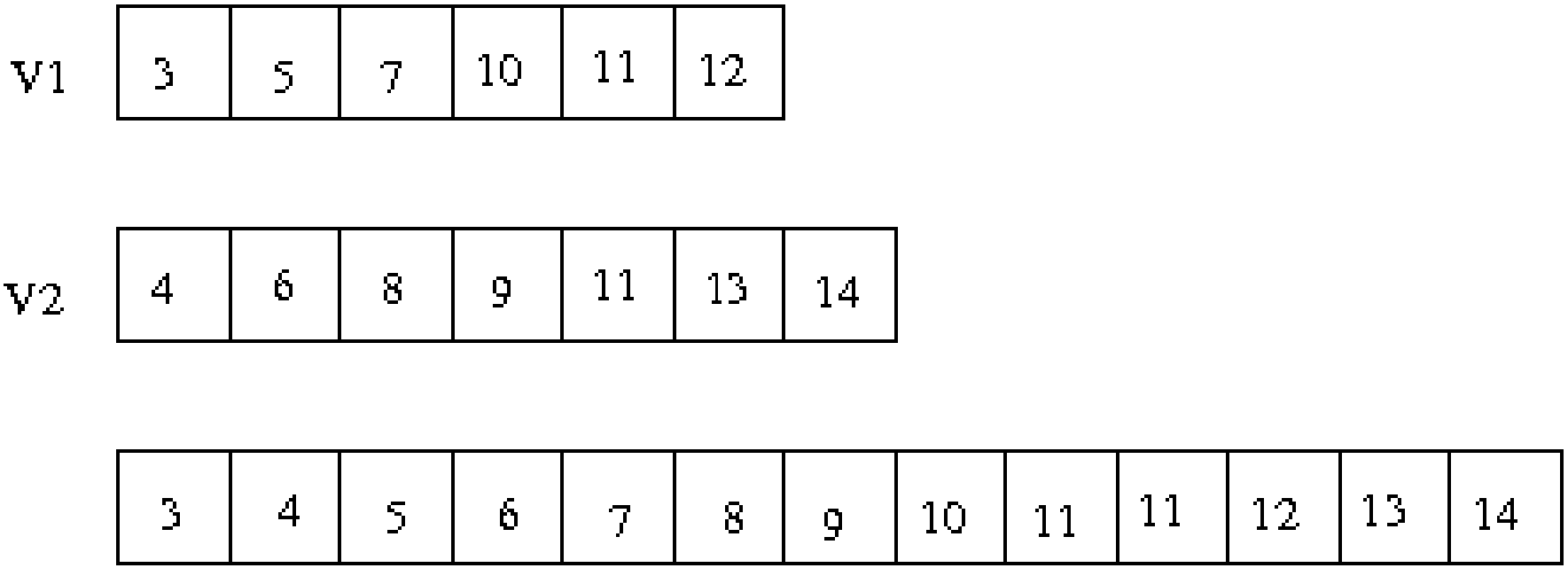
- O Merge-Sort é um algoritmo baseado na técnica dividir-e-conquistar.
- Neste caso temos que ordenar um vetor de tamanho n.
- Se $n = 1$ então o problema está resolvido pois o vetor está ordenado.
- Para $n > 1$ temos:
  - **Dividir:** Dividimos o vetor de tamanho n em dois sub-vetores de tamanho aproximadamente iguais (um de tamanho $\lceil n/2 \rceil$ e outro de tamanho $\lfloor n/2 \rfloor$).
  - Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores.
  - **Conquistar:** Com os dois sub-vetores ordenados, construímos um vetor de tamanho n ordenado.

Merge-Sort: Ordenação por intercalação

- O Merge-Sort é um algoritmo baseado na técnica dividir-e-conquistar.
- Neste caso temos que ordenar um vetor de tamanho n.
- Se $n = 1$ então o problema está resolvido pois o vetor está ordenado.
- Para $n > 1$ temos:
  - **Dividir:** Dividimos o vetor de tamanho n em dois sub-vetores de tamanho aproximadamente iguais (um de tamanho $\lceil n/2 \rceil$ e outro de tamanho $\lfloor n/2 \rfloor$).
  - Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores.
  - **Conquistar:** Com os dois sub-vetores ordenados, construímos um vetor de tamanho n ordenado.

Merge-Sort: Ordenação por intercalação

Conquistar: Dados dois vetores v_1 e v_2 ordenados, como obter um outro vetor ordenado contendo os elementos de v_1 e v_2?



Merge (Fusão)

- A ideia é executar um laço onde em cada iteração testamos quem é o menor elemento dentre $v_1[i]$ e $v_2[j]$, e copiamos este elemento para o novo vetor.
- Durante a execução deste laço podemos chegar em uma situação onde todos os elementos de um dos vetores (v_1 ou v_2) foram todos avaliados. Neste caso terminamos o laço e copiamos os elementos restantes do outro vetor.

Merge (Fusão)

- A ideia é executar um laço onde em cada iteração testamos quem é o menor elemento dentre $v_1[i]$ e $v_2[j]$, e copiamos este elemento para o novo vetor.
- Durante a execução deste laço podemos chegar em uma situação onde todos os elementos de um dos vetores (v_1 ou v_2) foram todos avaliados. Neste caso terminamos o laço e copiamos os elementos restantes do outro vetor.

Merge (Fusão)

Retorna um vetor ordenado que é a fusão dos vetores ordenados passados por parâmetro:

```
int *merge(int v1[], int t1, int v2[], int t2){
  int *v3 = malloc((t1+t2)*sizeof(int));
  int i=0, j=0, k=0; //indices de v1, v2 e v3

  while(i < t1 && j < t2){//Enquanto não terminar um dos vetores
    if(v1[i] < v2[j])
      v3[k++] = v1[i++];
    else
      v3[k++] = v2[j++];
  }

  while(i < t1) //copia resto de v1
    v3[k++] = v1[i++];

  while(j < t2) //copia resto de v2
    v3[k++] = v2[j++];

  return v3;
}
```

Merge (Fusão)

Retorna um vetor ordenado que é a fusão dos vetores ordenados passados por parâmetro:

```
int *merge(int v1[], int t1, int v2[], int t2){
  int *v3 = malloc((t1+t2)*sizeof(int));
  int i=0, j=0, k=0; //indices de v1, v2 e v3

  while(i < t1 && j < t2){//Enquanto não terminar um dos vetores
    if(v1[i] < v2[j])
      v3[k++] = v1[i++];
    else
      v3[k++] = v2[j++];
  }

  while(i < t1) //copia resto de v1
    v3[k++] = v1[i++];

  while(j < t2) //copia resto de v2
    v3[k++] = v2[j++];

  return v3;
}
```

Merge (Fusão)

Retorna um vetor ordenado que é a fusão dos vetores ordenados passados por parâmetro:

```
int *merge(int v1[], int t1, int v2[], int t2){
  int *v3 = malloc((t1+t2)*sizeof(int));
  int i=0, j=0, k=0; //indices de v1, v2 e v3

  while(i < t1 && j < t2){//Enquanto não terminar um dos vetores
    if(v1[i] < v2[j])
      v3[k++] = v1[i++];
    else
      v3[k++] = v2[j++];
  }

  while(i < t1) //copia resto de v1
    v3[k++] = v1[i++];

  while(j < t2) //copia resto de v2
    v3[k++] = v2[j++];

  return v3;
}
```

Merge (Fusão)

- A função descrita recebe dois vetores ordenados e devolve um terceiro contendo todos os elementos.
- Porém no merge-sort faremos a intercalação de sub-vetores usando sempre um mesmo vetor auxiliar.
  - Isto evitará a alocação de vários vetores durante as chamadas recursivas, melhorando a performance do algoritmo.
- Além disso os sub-vetores farão parte do vetor original a ser ordenado:
  - Teremos posições **ini, meio, fim** do vetor e devemos fazer a intercalação dos dois sub-vetores: um de **ini** até **meio**, e outro de **meio+1** até **fim**.
- A função utiliza o vetor auxiliar, que receberá o resultado da intercalação, e que no final é copiado para o vetor a ser ordenado.

Merge (Fusão)

- A função descrita recebe dois vetores ordenados e devolve um terceiro contendo todos os elementos.
- Porém no merge-sort faremos a intercalação de sub-vetores usando sempre um mesmo vetor auxiliar.
  - Isto evitará a alocação de vários vetores durante as chamadas recursivas, melhorando a performance do algoritmo.
- Além disso os sub-vetores farão parte do vetor original a ser ordenado:
  - Teremos posições **ini**, **meio**, **fim** do vetor e devemos fazer a intercalação dos dois sub-vetores: um de **ini** até **meio**, e outro de **meio+1** até **fim**.
- A função utiliza o vetor auxiliar, que receberá o resultado da intercalação, e que no final é copiado para o vetor a ser ordenado.

Merge (Fusão)

- A função descrita recebe dois vetores ordenados e devolve um terceiro contendo todos os elementos.
- Porém no merge-sort faremos a intercalação de sub-vetores usando sempre um mesmo vetor auxiliar.
  - Isto evitará a alocação de vários vetores durante as chamadas recursivas, melhorando a performance do algoritmo.
- Além disso os sub-vetores farão parte do vetor original a ser ordenado:
  - Teremos posições **ini, meio, fim** do vetor e devemos fazer a intercalação dos dois sub-vetores: um de **ini** até **meio**, e outro de **meio+1** até **fim**.
- A função utiliza o vetor auxiliar, que receberá o resultado da intercalação, e que no final é copiado para o vetor a ser ordenado.

Merge (Fusão)

- Abaixo está a função que faz intercalação de pedaços de **v**.
- No fim **v** estará ordenado entre as posições **ini** e **fim**:

```
void merge(int v[], int ini,  int meio, int fim, int aux[]){
  int i=ini,j=meio+1,k=0; //indices da metade inf, sup e aux respc.

  while(i<=meio && j<=fim){ //Enquanto não processou um sub-vetor inteiro.
    if(v[i] <= v[j])
      aux[k++] = v[i++];
    else
      aux[k++] = v[j++];
  }

  while(i<=meio) //copia resto do primeiro sub-vetor
    aux[k++] = v[i++];

  while(j<=fim)  //copia resto do segundo sub-vetor
    aux[k++] = v[j++];

  for(i=ini, k=0 ; i<= fim; i++, k++) //copia vetor ordenado aux para v
      v[i]=aux[k];
}
```

Merge (Fusão)

- Abaixo está a função que faz intercalação de pedaços de **v**.
- No fim **v** estará ordenado entre as posições **ini** e **fim**:

```
void merge(int v[], int ini,  int meio, int fim, int aux[]){
  int i=ini,j=meio+1,k=0; //indices da metade inf, sup e aux respc.


  while(i<=meio && j<=fim){ //Enquanto não processou um sub-vetor inteiro.
    if(v[i] <= v[j])
      aux[k++] = v[i++];
    else
      aux[k++] = v[j++];
  }


  while(i<=meio) //copia resto do primeiro sub-vetor
    aux[k++] = v[i++];

  while(j<=fim)  //copia resto do segundo sub-vetor
    aux[k++] = v[j++];

  for(i=ini, k=0 ; i<= fim; i++, k++) //copia vetor ordenado aux para v
      v[i]=aux[k];
}
```

Merge (Fusão)

- Abaixo está a função que faz intercalação de pedaços de **v**.
- No fim **v** estará ordenado entre as posições **ini** e **fim**:

```
void merge(int v[], int ini,  int meio, int fim, int aux[]){
  int i=ini,j=meio+1,k=0; //indices da metade inf, sup e aux respc.


  while(i<=meio && j<=fim){ //Enquanto não processou um sub-vetor inteiro.
    if(v[i] <= v[j])
      aux[k++] = v[i++];
    else
      aux[k++] = v[j++];
  }


  while(i<=meio) //copia resto do primeiro sub-vetor
    aux[k++] = v[i++];

  while(j<=fim)  //copia resto do segundo sub-vetor
    aux[k++] = v[j++];

  for(i=ini, k=0 ; i<= fim; i++, k++) //copia vetor ordenado aux para v
      v[i]=aux[k];
}
```

Merge-Sort

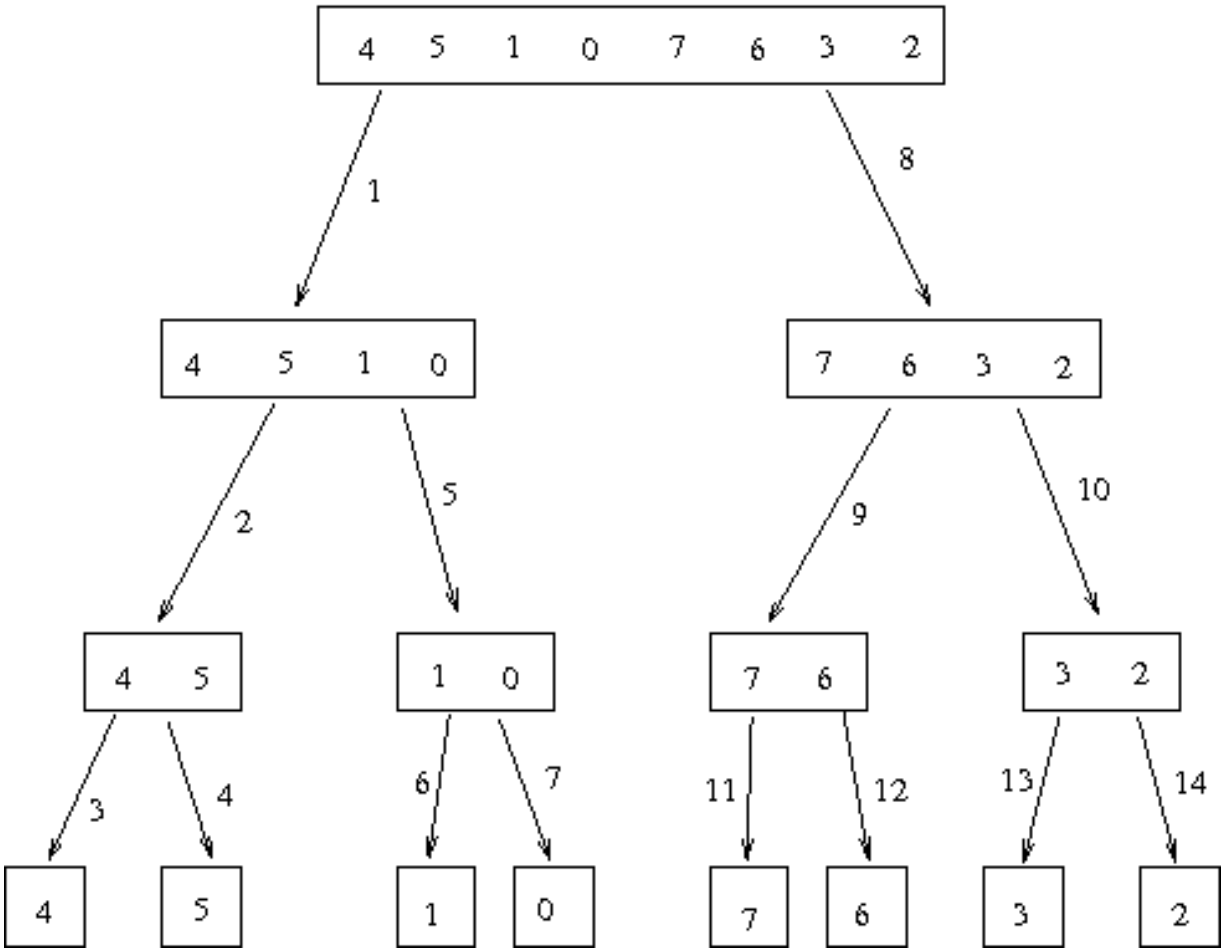
- O merge-sort resolve de forma recursiva dois sub-problemas, cada um contendo uma metade do vetor original.
- **No caso base** temos apenas um elemento e nada precisa ser feito.
- **No caso geral** resolvemos a ordenação dos sub-vetores recursivamente e depois chamamos a função **merge** para obter o vetor inteiro ordenado.

Merge-Sort

```
void mergeSort(int v[],int ini, int fim, int aux[]){
  int meio = (fim+ini)/2;
  if(ini < fim){    //Se tiver pelo menos 2 elementos então ordena
    mergeSort(v,ini,meio,aux);
    mergeSort(v,meio+1,fim,aux);
    merge(v,ini,meio,fim,aux);
  }
}
```

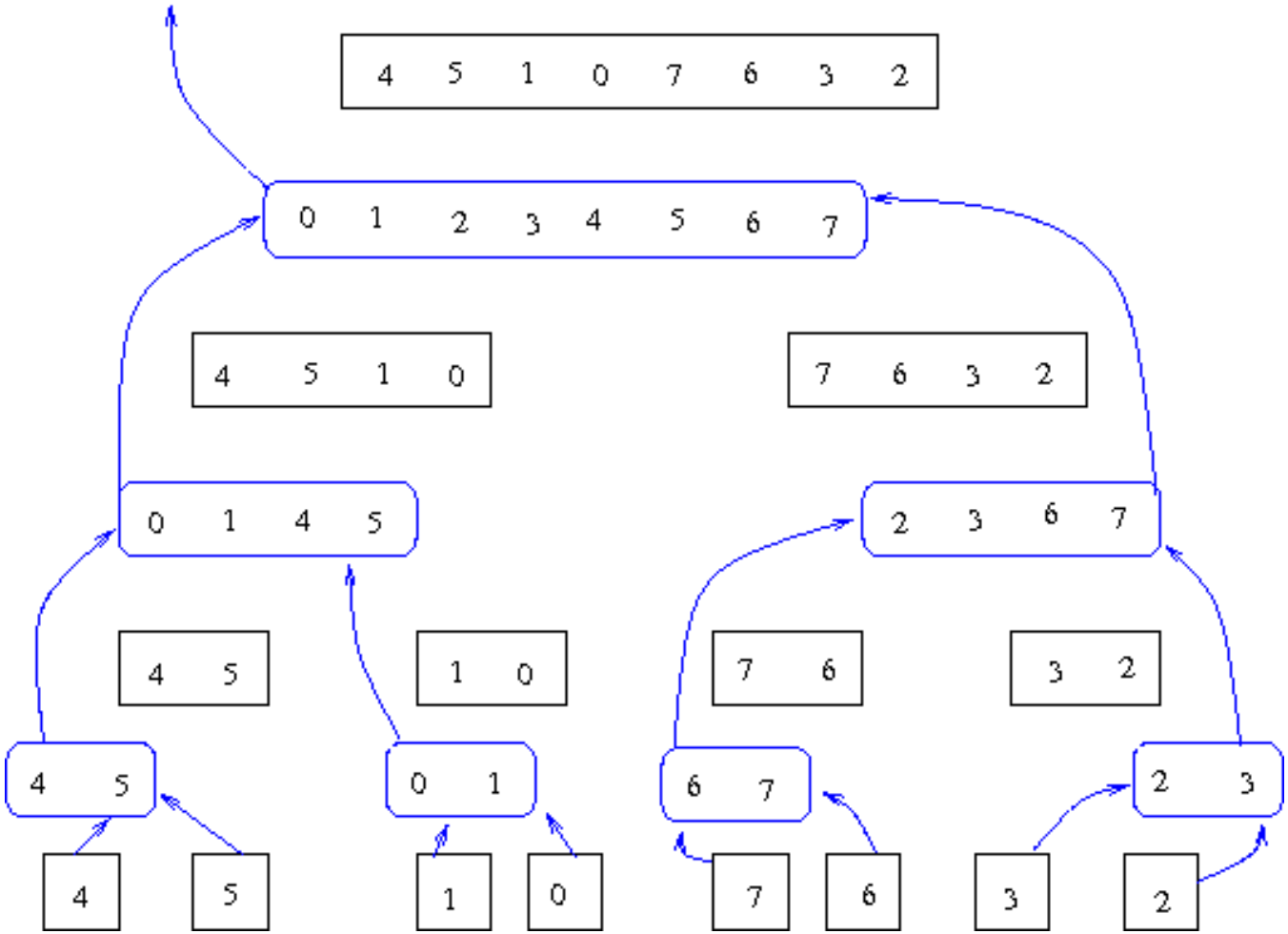
Merge-Sort

Abaixo temos um exemplo com a ordem de execução das chamadas recursivas.



Merge-Sort

Abaixo temos o retorno do exemplo anterior.



Merge-Sort: Exemplo de uso

- Note que só criamos 2 vetores, **v** a ser ordenado e **aux** do mesmo tamanho de **v**.
- Somente estes dois vetores existirão durante todas as chamadas recursivas.

```
#include "stdio.h"
#include <stdlib.h>

void merge(int v[], int ini,  int meio, int fim, int aux[]);
void mergeSort(int v[],int ini, int fim, int aux[]);


int main(){
    int v[]={12,90, 47, -9, 78, 45, 78, 3323, 1, 2, 34, 20};
    int aux[12];
    int i;
    mergeSort(v, 0, 11, aux);
    for(i=0; i<12; i++)
       printf("\n %d",v[i]);
}
```

Exercícios

1. Mostre passo a passo a execução da função merge considerando dois sub-vetores: $(3, 5, 7, 10, 11, 12)$ e $(4, 6, 8, 9, 11, 13, 14)$.
2. Faça uma execução Passo-a-Passo do Merge-Sort para o vetor: $(30, 45, 21, 20, 6, 715, 100, 65, 33)$.
3. Reescreva o algoritmo Merge-Sort para que este passe a ordenar um vetor em ordem decrescente.
4. Considere o seguinte problema: Temos como entrada um vetor de inteiros v (não necessariamente ordenado), e um inteiro x. Desenvolva um algoritmo que determina se há dois números em v cuja soma seja x. Tente fazer o algoritmo o mais eficiente possível. Utilize um dos algoritmos de ordenação na sua solução.