

Unidade IV:

Ordenação Parcial



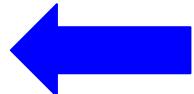
PUC Minas

Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

Agenda

- Introdução
- Seleção Parcial
- Inserção Parcial
- Quicksort Parcial
- Heapsort Parcial

Agenda

- **Introdução** 

- Seleção Parcial

- Inserção Parcial

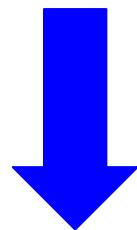
- Quicksort Parcial

- Heapsort Parcial

Introdução

- Dado um *array* com n elementos, desejamos que os k primeiros (menores) fiquem ordenados
- Seja $n = 16$ e $k = 3$

7	10	11	3	16	6	14	4	9	13	5	2	15	8	1	12
---	----	----	---	----	---	----	---	---	----	---	---	----	---	---	----



Ordenação parcial

1	2	3	11	16	6	14	4	9	13	5	10	15	8	7	12
---	---	---	----	----	---	----	---	---	----	---	----	----	---	---	----

Introdução

- Quando $k = 1$, o problema se reduz a encontrar o mínimo (ou o máximo) de um conjunto de elementos
- Quando $k = n$ caímos no problema clássico de ordenação
- Normalmente, $n \gg k$

Exemplo de Aplicação

Google

 Bing

yahoo!


mercado
livre

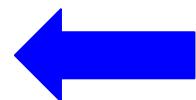
 amazon



 buscapé

Agenda

- Introdução
- **Seleção Parcial**
- Inserção Parcial
- Quicksort Parcial
- Heapsort Parcial



Seleção Parcial: Adaptação

- Selecionamos apenas os k menores

Seleção Parcial: Código em C-like

//Original

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

//Parcial

```
for (int i = 0; i < k; i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Exercício (1)

- Faça a análise de Complexidade do Seleção Parcial

Exercício (1)

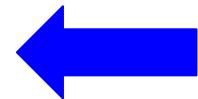
- Faça a análise de Complexidade do Seleção Parcial

$$C(n) = kn - \frac{k^2}{2} - \frac{k}{2}$$

$$M(n) = 3k$$

Agenda

- Introdução
- Seleção Parcial
- Inserção Parcial
- Quicksort Parcial
- Heapsort Parcial



Inserção Parcial: Adaptação

- Os k primeiros elementos são ordenados normalmente
- Para os demais elementos, se algum for menor que o valor da k-ésima posição, ele será inserido no conjunto já ordenado. Caso contrário, descartado
- Exemplo ($n = 16$ e $k = 4$)



0	1	2	3	4	6	14	4	9	13	5	10	15	8	7	0
---	---	---	---	---	---	----	---	---	----	---	----	----	---	---	---

Inserção Parcial: Algoritmo em C-like

```
//Original
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
```

```
//Parcial
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = (i < k) ? i - 1 : k - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
```

Inserção Parcial: Algoritmo em C-like

```
//Original
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
```

Observação: Neste caso, podemos ter a perda de elementos “desnecessários”

```
//Parcial
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = (i < k) ? i - 1 : k - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
```

Exercício (2)

- Faça a análise de Complexidade do Inserção Parcial

Exercício (2)

- Faça a análise de Complexidade do Inserção Parcial

- No anel mais interno, na i -ésima iteração o valor de C_i é:

Melhor caso : $C_i(n) = 1$

Pior caso : $C_i(n) = i$

Caso médio : $C_i(n) = \frac{1}{i}(1 + 2 + \dots + i) = \frac{i+1}{2}$

- Assumindo que todas as permutações de n são igualmente prováveis, o número de comparações é:

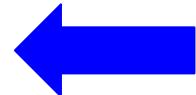
Melhor caso : $C(n) = (1 + 1 + \dots + 1) = n - 1$

Pior caso : $C(n) = (2 + 3 + \dots + k + (k + 1)(n - k))$
 $= kn + n - \frac{k^2}{2} - \frac{k}{2} - 1$

Caso médio : $C(n) = \frac{1}{2}(3 + 4 + \dots + k + 1 + (k + 1)(n - k))$
 $= \frac{kn}{2} + \frac{n}{2} - \frac{k^2}{4} + \frac{k}{4} - 1$

Agenda

- Introdução
- Seleção Parcial
- Inserção Parcial
- **Quicksort Parcial**
- Heapsort Parcial



Quicksort Parcial: Modificação

- Assim como o Quicksort, sua versão parcial é o algoritmo de ordenação parcial mais rápido em várias situações
- Basta abandonar a partição à direita quando a da esquerda tiver k ou mais itens

Quicksort Parcial: Algoritmo em C-like

//Original

```
void quicksort(int esq, int dir) {
    int i = esq,
        j = dir,
        pivo = array[(esq+dir)/2];
    while (i <= j) {
        while (array[i] < pivo)
            i++;
        while (array[j] > pivo)
            j--;
        if (i <= j)
            { swap(i, j); i++; j--; }
    }
    if (esq < j)
        quicksort(esq, j);
    if (i < dir)
        quicksort(i, dir);
}
```

//Parcial, sendo k variável da classe

```
void quicksort(int esq, int dir) {
    int i = esq,
        j = dir,
        pivo = array[(esq+dir)/2];
    while (i <= j) {
        while (array[i] < pivo)
            i++;
        while (array[j] > pivo)
            j--;
        if (i <= j)
            { swap(i, j); i++; j--; }
    }
    if (esq < j)
        quicksort(esq, j);
    if (i < k && i < dir)
        quicksort(i, dir);
}
```

Quicksort Parcial: Análise de Complexidade

- A análise do Quicksort é difícil
- O comportamento é muito sensível à escolha do pivô
- Podendo cair no melhor caso $\Theta(k \times \lg(k))$
- Ou em algum valor entre o melhor caso e $\Theta(n \times \lg(n))$

Agenda

- Introdução
- Seleção Parcial
- Inserção Parcial
- Quicksort Parcial
- **Heapsort Parcial** 

Heapsort Parcial (V1): Modificação

- Utiliza um *heap* para informar o menor item do conjunto
- Na primeira iteração, o menor item que está em **A[1]** (raiz do *heap*) é trocado com o item que está em **A[n]**
- Em seguida, o *heap* é refeito
- Novamente, o menor está em **A[1]**, troque-o com **A[n-1]**
- Repita as duas últimas operações até que o k -ésimo menor seja trocado com **A[n - k]**
- Ao final, os k menores estão nas k últimas posições do vetor **A**

Heapsort Parcial (V1): Algoritmo em C-like

```
void heapsortParcial(int *A, int n, int k) {  
    int esq = 1, dir, aux = 0;  
    construir (A, n); /* construir o heap */  
    dir = n;  
    while (aux < k) { /* ordena o vetor */  
        x = A[1] ;  
        A[1] = A[n - aux] ;  
        A[n - aux] = x;  
        dir--; aux++;  
        refaz(esq, dir , A) ;  
    }  
}
```

Heapsort Parcial (V1): Análise de Complexidade

- construir um heap com custo $O(n)$
- O procedimento refaz tem custo $O(lg(n))$
- O procedimento heapParcial chama o refaz k vezes
- Logo, o algoritmo apresenta a complexidade:

$$O(n + k \log n) = \begin{cases} O(n) & \text{se } k \leq \frac{n}{\log n} \\ O(k \log n) & \text{se } k > \frac{n}{\log n} \end{cases}$$

Heapsort Parcial (V1): Análise de Complexidade

- construir um heap com custo $O(n)$
- O procedimento refaz tem custo $O(lg(n))$
- O procedimento heapParcial chama o refaz k vezes
- Logo, o algoritmo apresenta a complexidade:

$$O(n + k \log n) = \begin{cases} O(n) & \text{se } k \leq \frac{n}{\log n} \\ O(k \log n) & \text{se } k > \frac{n}{\log n} \end{cases}$$

Heapsort Parcial (V2): Modificação

- Primeira etapa: Construa o *heap* invertido usando apenas com os k primeiros elementos. Para cada um dos $(n-k)$ demais elementos, se ele for menor que a raiz do *heap*, troque-o com a raiz e reorganize o *heap*
- Segunda etapa: Igual à original, contudo, sendo que o *heap* tem tamanho k

Heapsort Parcial (V2): Algoritmo em C-like

//Original

```
void heapsort() {  
    //Construção do heap  
    for (int tam = 2; tam <= n; tam++)  
        construir(tam);  
  
    //Ordenação propriamente dita  
    int tam = n;  
    while (tam > 1){  
        swap(1, tam--);  
        reconstruir(tam);  
    }  
}
```

//Parcial, k é uma variável da classe
void heapsort() {

```
    //Construção do heap com os k primeiros elementos  
    for (int tam = 2; tam <= k; tam++)  
        construir(tam);
```

```
    //Para cada um dos (n-k) demais elementos, se ele for  
    //menor que a raiz, inserir do heap  
    for (int i = k + 1; i <= n; i++)  
        if (array[i] < array[1]) {  
            swap(i, 1);    reconstruir(k);  
        }
```

```
    //Ordenação propriamente dita  
    int tam = k;  
    while (tam > 1){  
        swap(1, tam--);  reconstruir(tam);  
    }  
}
```

Heapsort Parcial (V2): Análise de Complexidade

- Construa o *heap* invertido usando apenas com os k primeiros elementos
 - Melhor caso: $\Theta(k)$
 - Pior caso: $\Theta(k \times \lg k)$
- Para cada um dos $(n-k)$ demais elementos ...
 - Melhor caso: $\Theta(n-k)$
 - Pior caso: $\Theta(n \times \lg k)$
- Segunda etapa: Igual à original, contudo, sendo que o *heap* tem tamanho k
 - Todos os casos: $\Theta(k \times \lg k)$
- Custo final
 - Melhor caso: $\Theta(k) + \Theta(n-k) + \Theta(k \times \lg k) = \Theta(n-k)$
 - Pior caso: $\Theta(k \times \lg k) + \Theta(n \times \lg k) + \Theta(k \times \lg k) = \Theta(n \times \lg k)$