AED1 - Aula 27
Problemas da seleção e da contagem de inversões

Uma das ideias centrais em algoritmos é que
- abordagens usadas para resolver um problema
    - podem ser bem sucedidas quando aplicadas/adaptadas
        - para problemas diferentes.

Nesta aula veremos como usar o que aprendemos
- ao estudar algoritmos para o problema da ordenação
    - para projetar algoritmos para dois problemas relacionados.

Problema da seleção

Definições:
- a ordem de um elemento é uma medida da grandeza dele
    - em relação aos seus pares.
- Assim, se a ordem de um elemento é k
    - então existem k elementos de valor menor que o dele.
- Dado um vetor v de tamanho n e um inteiro k em [0, n)
    - no problema da seleção queremos o valor do elemento de ordem k

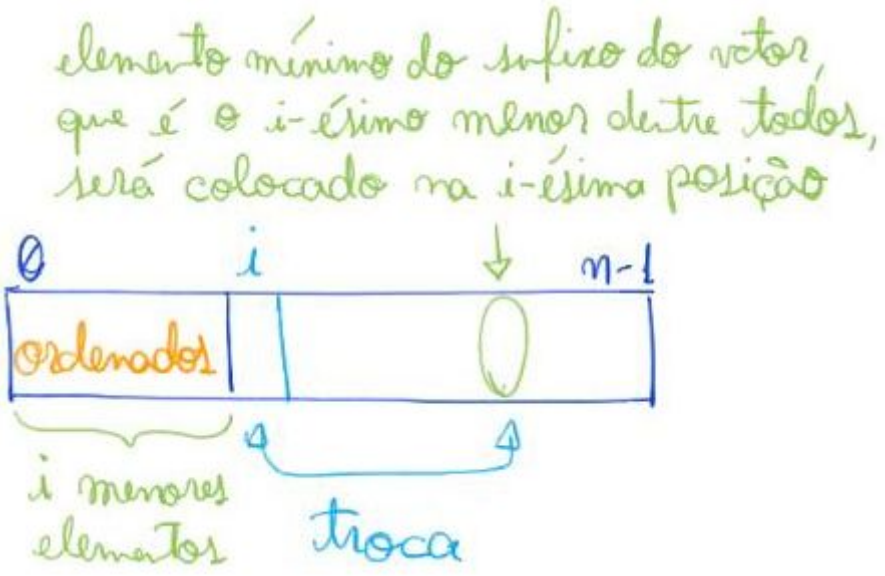
Exemplos:
- 3 2 5 4 1 e k = 3
    - Elemento de ordem 3 é 4
- 1 2 3 4 5 e k = 3
    - Elemento de ordem 3 é 4
- 5 4 3 2 1 e k = 3
    - Elemento de ordem 3 é 4

A resposta não muda nas diferentes permutações,
- pois a ordem de um elemento depende da comparação
    - de seu valor com os demais elementos,
- e não de sua posição no vetor.

Curiosidades:
- Na permutação ordenada do vetor v[0 .. n - 1]
    - o elemento de ordem k ocupa a k-ésima posição.
- Note que, podemos definir ordem começando em 0 ou em 1.
    - Escolhi usar ela começando em 0, para combinar com nossos vetores.
    - Assim, o elemento de ordem k ocupa a posição v[k] se v for ordenado.

- Perceba que o problema do mínimo e do máximo são casos particulares
    - do problema da seleção.
    - Mínimo corresponde ao elemento de ordem 0.
    - Máximo corresponde ao elemento de ordem n - 1.
- Observe que o problema da seleção é trivial se v estiver ordenado,
    - ou se ordenarmos ele.
        - Qual seria a eficiência dessa abordagem?
- Será que conseguimos resolver o problema sem usar esta abordagem?

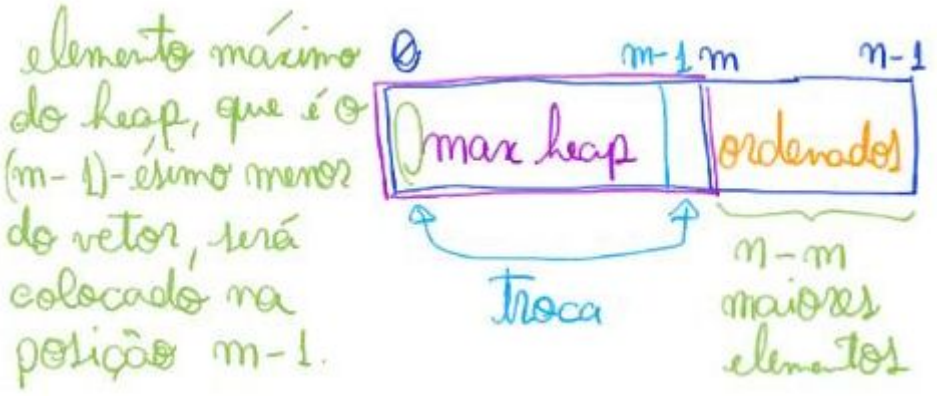
Algoritmo baseado na ideia do selectionSort:




```c
int selecao1(int v[], int n, int k)
{
   int i, j, ind_min, aux;
   for (i = 0; i <= k; i++)
   {
       ind_min = i;
       for (j = i + 1; j < n; j++)
           if (v[j] < v[ind_min])
               ind_min = j;
       troca(&v[i], &v[ind_min]);
   }
   return v[k];
}
```

- Invariante e corretude: v[0 .. i - 1] está ordenado e é <= que v[i .. n - 1]
- Eficiência de tempo: O(k n),
    - dado que fazemos k buscas lineares pelo mínimo
        - para encontrar o k-ésimo elemento.
- Eficiência de espaço: O(1) espaço adicional.

Algoritmo baseado na ideia do heapSort:



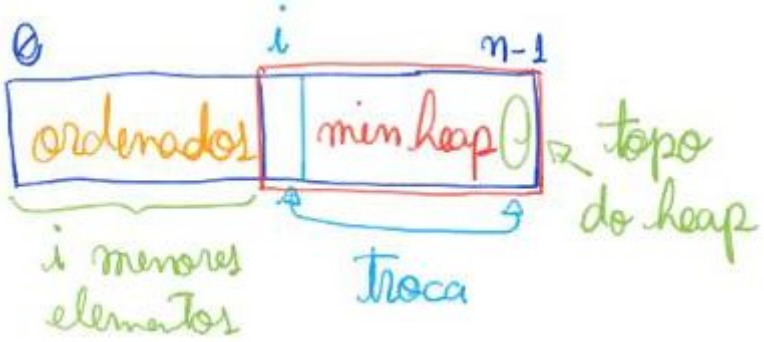

```
int selecao2(int v[], int n, int k)
{
   int i, m = n;
   for (i = n / 2; i >= 0; i--)
       desceHeap(v, n, i);
   for (m = n - 1; m >= k; m--)
   {
       troca(&v[0], &v[m]);
       desceHeap(v, m, 0);
   }
   return v[k];
}
```

- Invariante e corretude:
  - v[m + 1 .. n - 1] está ordenado e é >= que v[0 .. m],
    - que é um heap de máximo
- Eficiência de tempo: O(n + (n - k) lg n),
  - sendo o primeiro O(n) gasto para construir o heap,
  - e (n - k) o número de remoções do heap de máximo,
    - até encontrar o k-ésimo elemento.
  - Note que, se (n - k) <= n / lg n
    - a eficiência do algoritmo é linear em n, i.e., O(n).
- Eficiência de espaço: O(1) espaço adicional.

Quizz:

- Como podemos melhorar os algoritmos anteriores,
  - usando como base a comparação de k com n?
- Realizar a melhoria em selecao1 é simples,
  - mas aplicar a mesma ideia em selecao2

        - apresenta complicações envolvendo o heap.



Curiosidade:

- Futuramente veremos que
    - usando a técnica de divisão e conquista,
        - que é a base da busca binária,
    - junto de escolhas aleatórias,
    - é possível obter soluções mais eficientes para esse problema.

Problema da Contagem de Inversões

Definição:

- Uma inversão corresponde a um par de elementos v[i] e v[j]
    - tal que i < j e v[i] > v[j]
- Dado um vetor v de tamanho n,
    - queremos saber quantas inversões existem em v

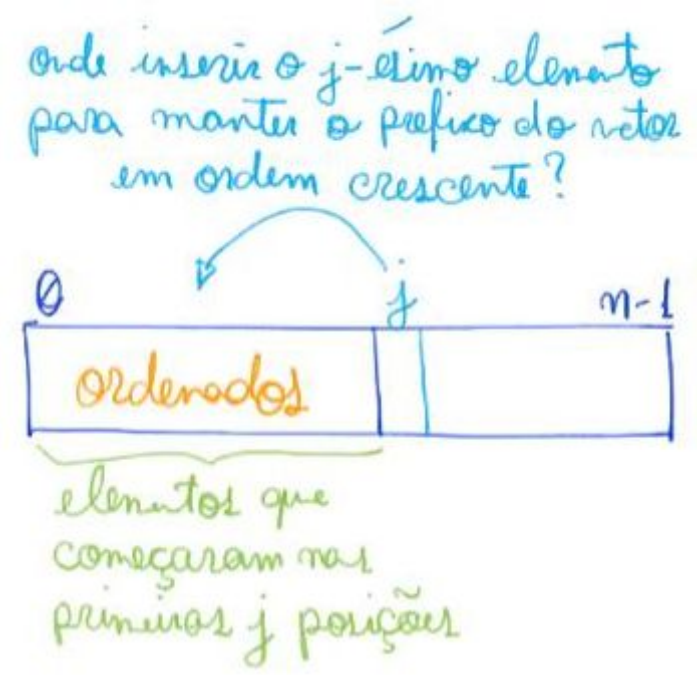
Exemplos:

- 3 2 5 4 1
    - 3 está invertido com 2 e 1
    - 2 está invertido com 1
    - 5 está invertido com 4 e 1
    - 4 está invertido com 1
    - Total de inversões = 2 + 1 + 2 + 1 = 6
- 1 2 3 4 5
    - Total de inversões = 0
- 5 4 3 2 1
    - 5 está invertido com 4, 3, 2 e 1
    - 4 está invertido com 3, 2 e 1
    - 3 está invertido com 2 e 1
    - 2 está invertido com 1
    - Total de inversões = 4 + 3 + 2 + 1

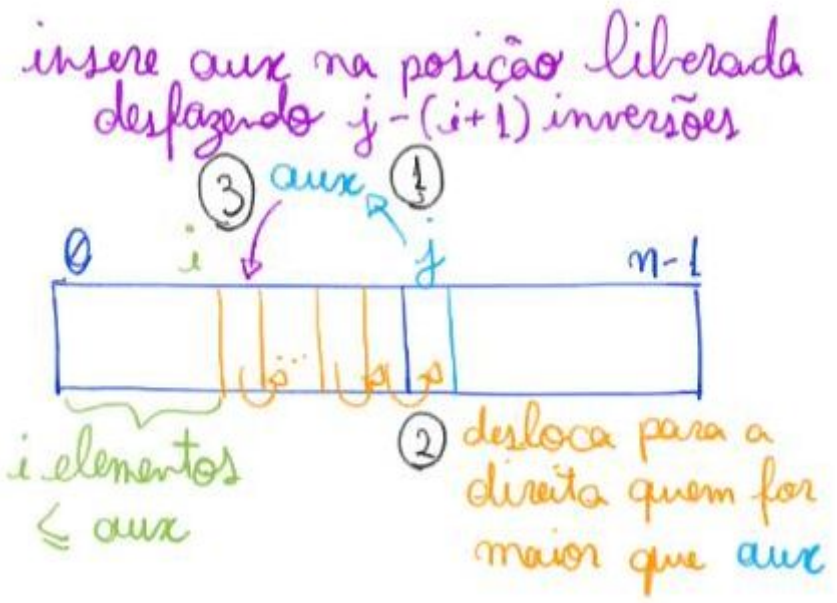
Curiosidades:

- Número mínimo de inversões = 0
    - ocorre quando o vetor está em ordem crescente
- Número máximo de inversões = (n escolhe 2) = n(n - 1)/2
    - o valor (n escolhe 2) corresponde a todo par ser uma inversão
    - ocorre quando o vetor está em ordem decrescente
- Assim, podemos pensar no número de inversões
    - como uma medida da desordem dos elementos de um vetor.

Algoritmo baseado na ideia do insertionSort:




```
unsigned long long contarInversoes1(int v[], int n)
{
   int i, j, aux;
   unsigned long long num_inv = 0;
   for (j = 1; j < n; j++)
   {
       aux = v[j];
       for (i = j - 1; i >= 0 && aux < v[i]; i--)
       {
           v[i + 1] = v[i];
           // num_inv++;
       }
       num_inv += j - 1 - i;
       v[i + 1] = aux; /* por que i+1? */
   }
   return num_inv;
}
```



- Invariante e corretude:
  - v[0 .. j - 1] está ordenado
  - num_inv = número de inversões envolvendo apenas elementos do subvetor v[0 .. j - 1] original
- Eficiência de tempo:
  - O(n^2) no pior caso
  - O(n) no melhor caso
- Eficiência de espaço: O(1) espaço adicional

Algoritmo baseado na ideia do bubbleSort:

```
unsigned long long contarInversoes2(int v[], int n)
{
   int j, i, aux, ut, l;
   unsigned long long num_inv = 0;
   l = n;
   for (j = 0; j < n; j++)
   {
       ut = 0;
       for (i = 1; i < l; i++)
           if (v[i - 1] > v[i])
           {
               aux = v[i - 1];
               v[i - 1] = v[i];
               v[i] = aux;
               ut = i;
               num_inv++;
           }
       l = ut;
   }
   return num_inv;
```

```
}
```

- Invariante e corretude:
    - v[l .. n - 1] está ordenado e é >= v[0 .. l - 1]
    - num_inv = número de inversões desfeitas até o momento
- Eficiência de tempo:
    - O(n^2) no pior caso
    - O(n) no melhor caso
- Eficiência de espaço: O(1) espaço adicional

Quizz:

- Todas as nossas adaptações de algoritmos para contagem de inversões
    - são de algoritmos de ordenação estável. Será coincidência?

Curiosidade:

- Futuramente veremos que usando a técnica de divisão e conquista
    - é possível obter soluções mais eficientes para esse problema.