BC1424
Algoritmos e Estruturas de Dados I

Aula 12:
Métodos eficientes de ordenação
(MergeSort)

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

1Q-2015

Ordenação

- Ordenar corresponde ao **processo de re-arranjar um conjunto de objetos** em ordem ascendente ou descendente (geralmente considerado no primeiro passo para resolver um problema prático).
- O objetivo principal da ordenação **é facilitar a recuperação posterior** de itens do conjunto ordenado.
- As ordens mais utilizadas são a numérica e lexicográfica.
- **Diversos algoritmos** de ordenação serão estudados e implementados...

Da última aula

Selection Sort

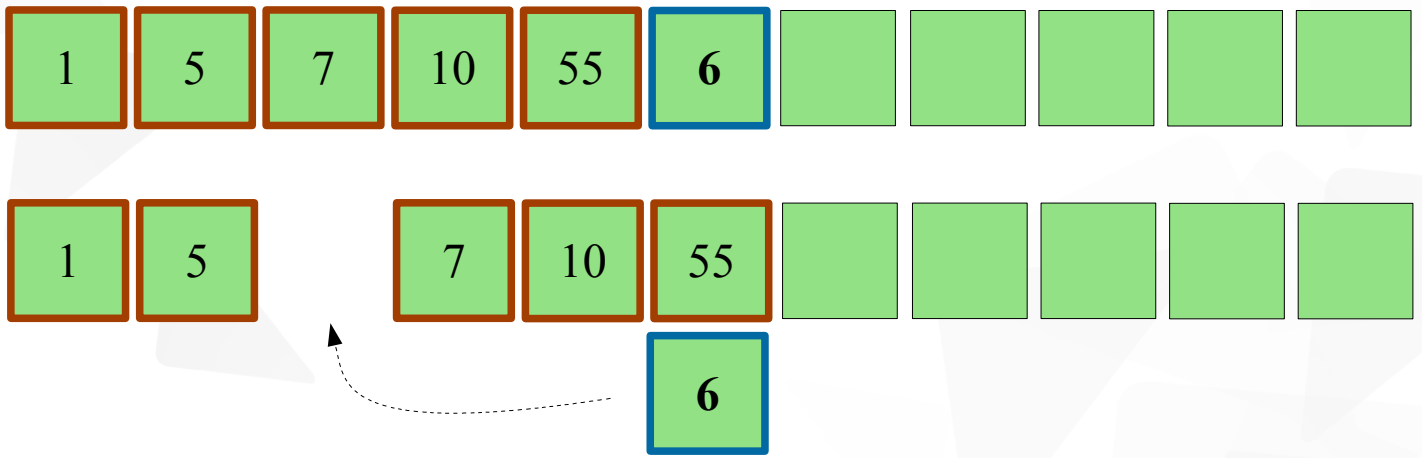
- Procura-se (seleciona-se) o primeiro menor e se posiciona na primeira posição
- Procura-se (seleciona-se) o segundo menor e se posiciona na segunda posição
- ...

Complexidade computacional, em termos de comparação entre elementos:
- Pior caso = $O(n^2)$
- melhor caso = $O(n^2)$

Insertion Sort

Método preferido dos jogadores de cartas

Em cada passo, a partir do i=2, o I-ésimo elemento da sequência fonte é apanhado e transferido para a sequência destino, sendo inserido no seu lugar apropriado.

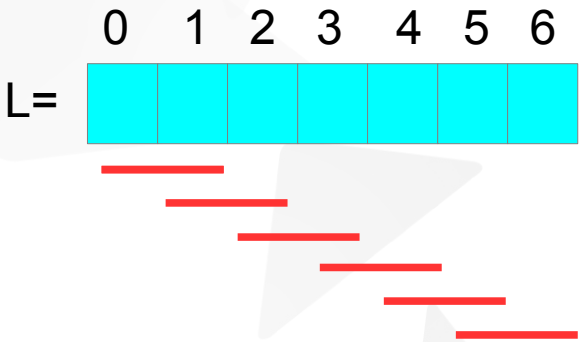


Complexidade computacional, em termos de comparação entre elementos:
- Pior caso = $O(n^2)$
- melhor caso = $O(n)$

Este algoritmo é o mais apropriado quando o vetor estiver semi-ordenado.

Bubble Sort

- O principio do bolha é a troca de valores entre posições consecutivas fazendo com que os valores mais altos “borbulhem” para o final do vetor.

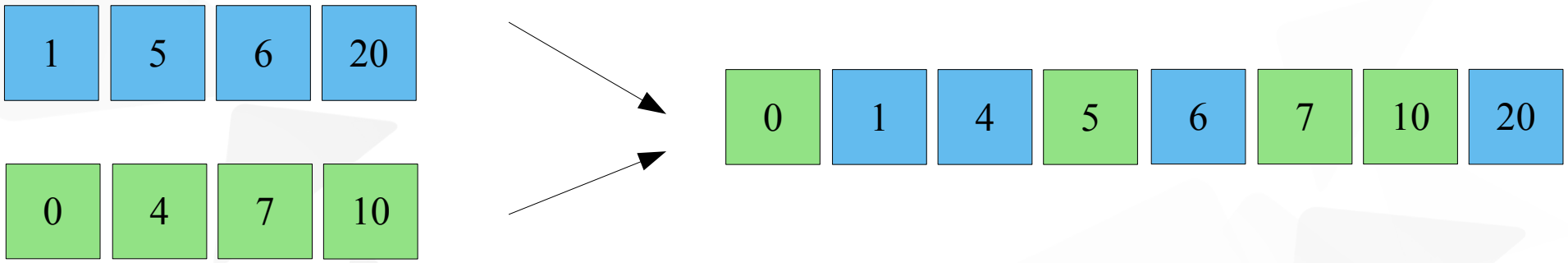


Complexidade computacional, em termos de comparação entre elementos:
- Pior caso = $O(n^2)$
- melhor caso = $O(n^2)$

Existem variações em que o melhor caso é $O(n)$

Merge Sort

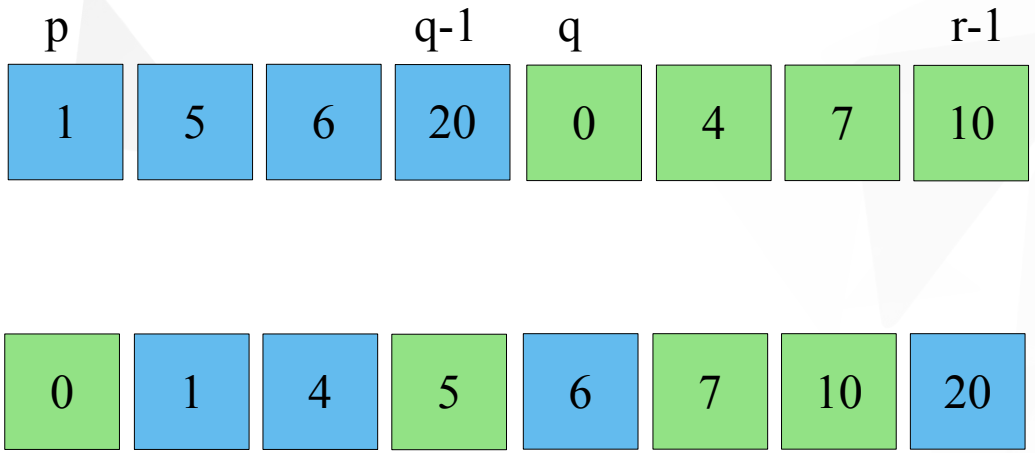
Intercala



Intercala

Antes de tratar do problema de ordenação, é preciso resolver um problema auxiliar:

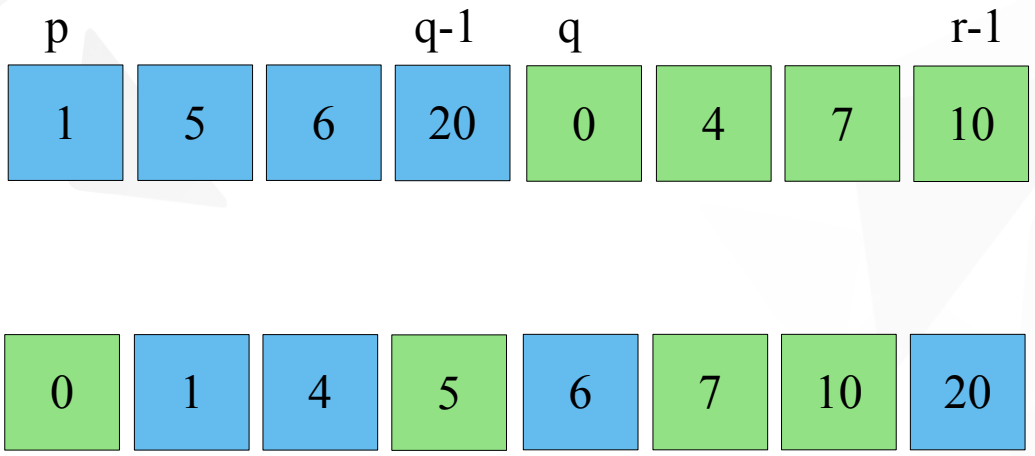
- Dados 2 vetores crescentes: `v[p..q-1]` e `v[q..r-1]` rearranjar `v[p..r-1]` em ordem crescente.
- Podemos dizer que o problama consiste em "**intercalar**" os dois vetores



Intercala

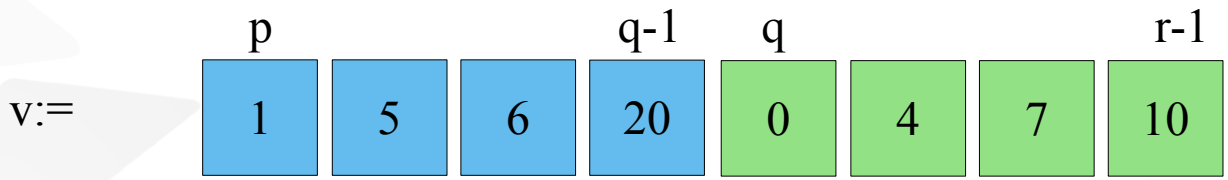
O problema pode ser resolvido em tempo proporcional ao quadrado de `r-p`, basta ordenar segundo algum algoritmo visto na última aula.

- Aqui ignoramos o fato de que as duas “metades” estão ordenadas.
- Podemos resolver de forma mais eficiente?



Intercala

Podemos usar um vetor auxiliar, digamos w, do mesmo tamanho que `v[p..r-1]` para resolver o problema



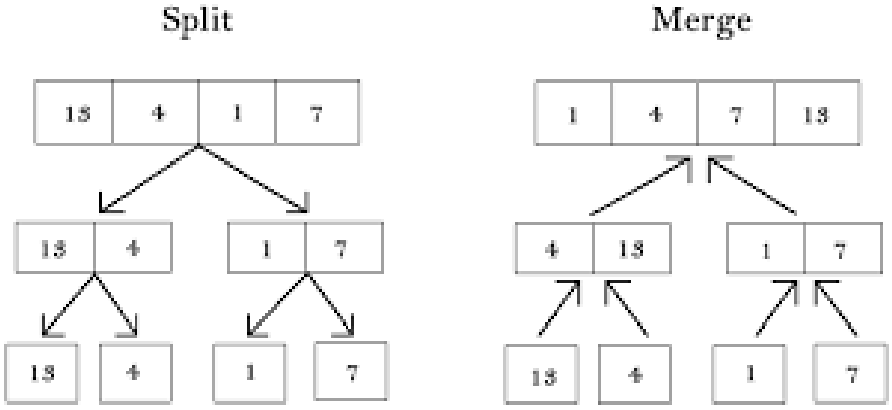
Faça um programa ***O(r-p)*** que intercale dos sub-vetores ordenados (na sua forma crescente) e devolva o vetor completo ordenado.

```c
void Intercala(int p, int q, int r, int v[]) {
    int i, j, k;
    int *w = (int *)malloc( (r-p)*sizeof(int) );

    i = p;
    j = q;
    k = 0;

    while (i<q && j<r) {
        if (v[i]<v[j])
            w[k++] = v[i++];
        else
            w[k++] = v[j++];
    }

    while(i<q)
        w[k++] = v[i++];

    while(j<r)
        w[k++] = v[j++];

    for (i=p; i<r; i++)
        v[i] = w[i-p];

    free(w);
}
```

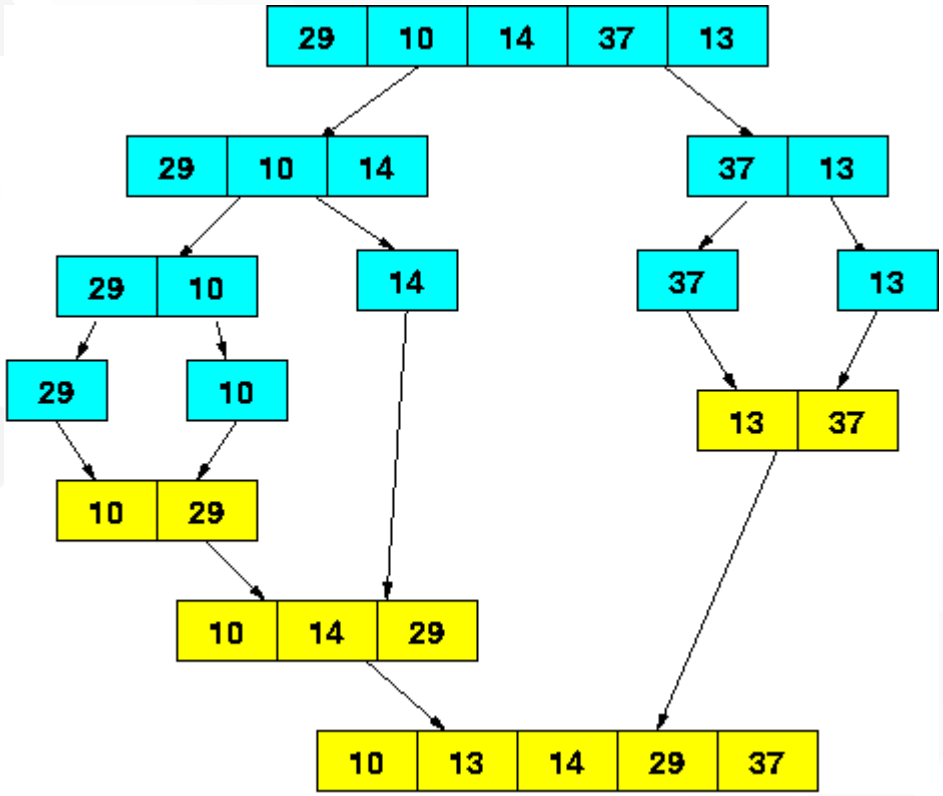
```c
void Intercala(int p, int q, int r, int v[]) {
    int i, j, k;
    int *w = (int *)malloc( (r-p)*sizeof(int) );

    i = p;
    j = q;
    k = 0;

    while (i<q && j<r) {
        if (v[i]<v[j])
            w[k++] = v[i++];
        else
            w[k++] = v[j++];
    }

    while(i<q)
        w[k++] = v[i++];

    while(j<r)
        w[k++] = v[j++];

    for (i=p; i<r; i++)
        v[i] = w[i-p];

    free(w);
}
```

Complexidade computacional, baseado em comparações, no pior caso = ***r-p***

Merge sort



Merge sort



Animação: http://www.sorting-algorithms.com/merge-sort

Merge sort

Faça uma função recursiva que permita ordenar um vetor `v[p..r-1]` na sua forma crescente.

Considere a abordagem do Merge Sort.

Assinatura:

```
void MergeSort (int p, int r, int v[])
```

Merge sort

```
void MergeSort (int p, int r, int v[]) {
    if (p<r-1) {
        int q = (p+r)/2;
        MergeSort(p, q, v);
        MergeSort(q, r, v);
        Intercala(p, q, r, v);
    }
}
```

Complexidade computacional, baseado em comparações, O (n lg(n))

onde n = r-p;

Merge Sort

Na primeira rodada, o problema original de ordenar v[0..n-1] é reduzido a dois outros:

Ordenar

- `v[0..n/2-1]`
- `v[n/2..n-1]`

Na segunda rodada temos quatro problemas:

Ordenar

- `v[0..n/4-1]`
- `v[n/4..n/2-1]`
- `v[n/2..3n/4-1]`
- `V[3n/4-n-1]`

O tempo total que intercala gasta em cada rodada é proporcional a n, logo, o consumo total é proporcional a n.lg(n)

Merge Sort

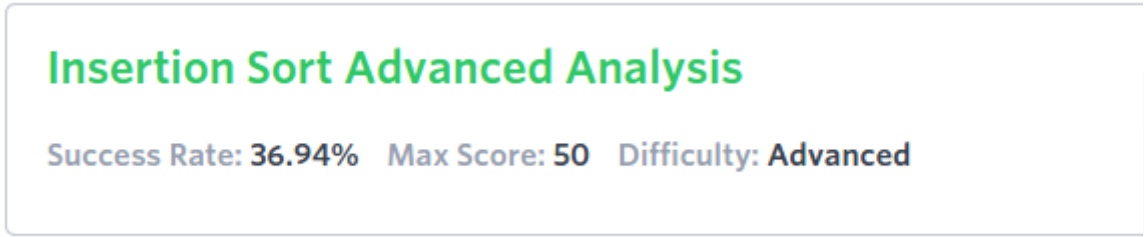
Merge Sort só é realmente mais rápido que os algoritmos vistos anteriormente, quando n é suficiente grande, uma vez que a constante de proporcionalidade na expressão “proporcional a” é maior no caso do Merge Sort.

Desafio 02 – opcional – envio até 01/04 (23h50-Tidia)

Para quem preferir incrementar sua nota final:

- Resolva o problema no HackerRank:
  Insertion Sort Advanced Analysis

- Sua solução deve passar por TODOS os testes.



Cada desafio vale +1/3 na média final da disciplina

Language: C

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

void insertionSort2 (int n, int v[]) {
    int i, j, aux;
    int shifts=0;

    for (i=1; i<n; i++) {
        aux = v[i];

        for (j=i-1; j>=0 && v[j]>aux ; j--) {
            v[j+1] = v[j];
            shifts++;
        }

        v[j+1] = aux;
    }
    printf("%d\n", shifts);
}

int main()
{
    int i, t, n;
    scanf("%d", &t);

    for (i=0; i<t; i++) {
        scanf("%d", &n);

        int v[n], e;
        for(e=0; e<n; e++) {
            scanf("%d", &v[e]);
        }

        insertionSort2(n, v);
    }

}
```

Solução incompleta!

# 0 ✔ 0s : Success (sample)	# 4 ✘ 3s : Terminated due to timeout	# 8 ✘ 3s : Terminated due to timeout
# 1 ✔ 0s : Success (sample)	# 5 ✔ 1.2s : Success	# 9 ✘ 3s : Terminated due to timeout
# 2 ✔ 0s : Success (sample)	# 6 ✔ 0.54s : Success	
# 3 ✔ 0s : Success	# 7 ✘ 3s : Terminated due to timeout	

Dica:
Use o algoritmo **MergeSort** para contar o número de trocas (shifts ou inversões)