AED2 - Aula 10
Ordenação por intercalação (mergesort)

Projeto de algoritmos por divisão e conquista

- Dividir: o problema é dividido em subproblemas menores do mesmo tipo.
- Conquistar: os subproblemas são resolvidos recursivamente, sendo que os subproblemas pequenos são caso base.
- Combinar: as soluções dos subproblemas são combinadas numa solução do problema original.

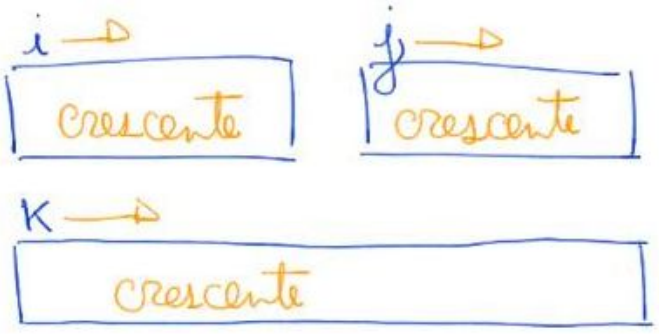
Ideia do mergesort

- Dividir o vetor a ser ordenado em dois subvetores,
  - cada um com metade do tamanho original.
- Ordenar cada subvetor recursivamente,
  - sendo que subvetores com 0 ou 1 elementos já estão ordenados.
- Intercalar (merge) os subvetores ordenados resultantes.

Algoritmo mergeSort recursivo

```
// ordena o vetor v[p .. r-1]
void mergeSortR(int v[], int p, int r)
{
    int m;
    if (r - p > 1)
    {
        m = (p + r) / 2;
        // m = p + (r - p) / 2;
        mergeSortR(v, p, m);
        mergeSortR(v, m, r);
        intercala1(v, p, m, r);
    }
}
```

Algoritmo de intercalação de subvetores ordenados




```c
// primeiro subvetor em v[p .. q-1], segundo subvetor em v[q .. r-1]
void intercala1(int v[], int p, int q, int r)
{
    int i, j, k, tam;
    i = p;
    j = q;
    k = 0;
    tam = r - p;
    int *w = malloc(tam * sizeof(int));
    while (i < q && j < r)
    {
        if (v[i] <= v[j])
            w[k++] = v[i++];
        else // v[i] > v[j]
            w[k++] = v[j++];
    }
    while (i < q)
        w[k++] = v[i++];
    while (j < r)
        w[k++] = v[j++];
    for (k = 0; k < tam; k++)
        v[p + k] = w[k];
    free(w);
}
```

Invariantes e corretude do intercala:

- no início de cada iteração temos
    - w[0 .. k - 1] contém os elementos de v[p .. i - 1] e v[q .. j - 1],
    - w[0 .. k - 1] está ordenado,

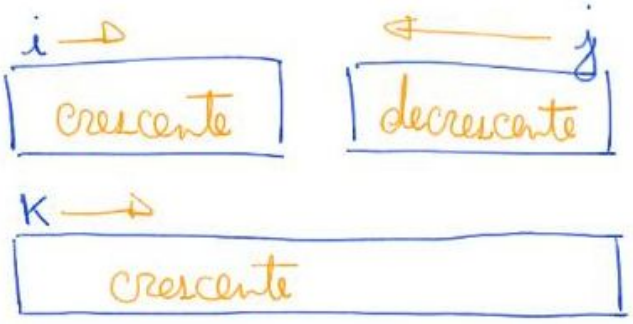
    - w[h] <= v[l] para 0 <= h < k e i <= l < q, i.e., w[0 .. k - 1] <= v[i .. q - 1],
    - w[h] <= v[l] para 0 <= h < k e j <= l < r, i.e., w[0 .. k - 1] <= v[j .. r - 1].

Eficiência de tempo do intercala:

- O número de operações é linear no tamanho do subvetor sendo intercalado,
    - ou seja, O(r - p).
- Para verificar isso, note que em cada iteração, de qualquer laço,
    - i ou j aumenta de 1.
- Como i começa em p e termina em q
    - e j começa em q e termina em r,
        - temos (q - p) + (r - q) = r - p iterações.

Curiosidade:

- Sedgewick propõe uma versão interessante do algoritmo de intercalação,
    - chamado intercalação com sentinelas.




```c
// primeiro subvetor em v[p .. q-1], segundo subvetor em v[q .. r-1]
void intercala2(int v[], int p, int q, int r)
{
    int i, j, k, *w;
    w = malloc((r - p) * sizeof(int));
    for (i = p; i < q; ++i)
        w[i - p] = v[i];
    for (j = q; j < r; ++j)
        w[(r - p - 1) - (j - q)] = v[j];
    i = 0;
    j = r - p - 1;
    for (k = p; k < r; ++k)
        if (w[i] <= w[j])
            v[k] = w[i++];
        else
            v[k] = w[j--];
```

```
    free(w);
}
```

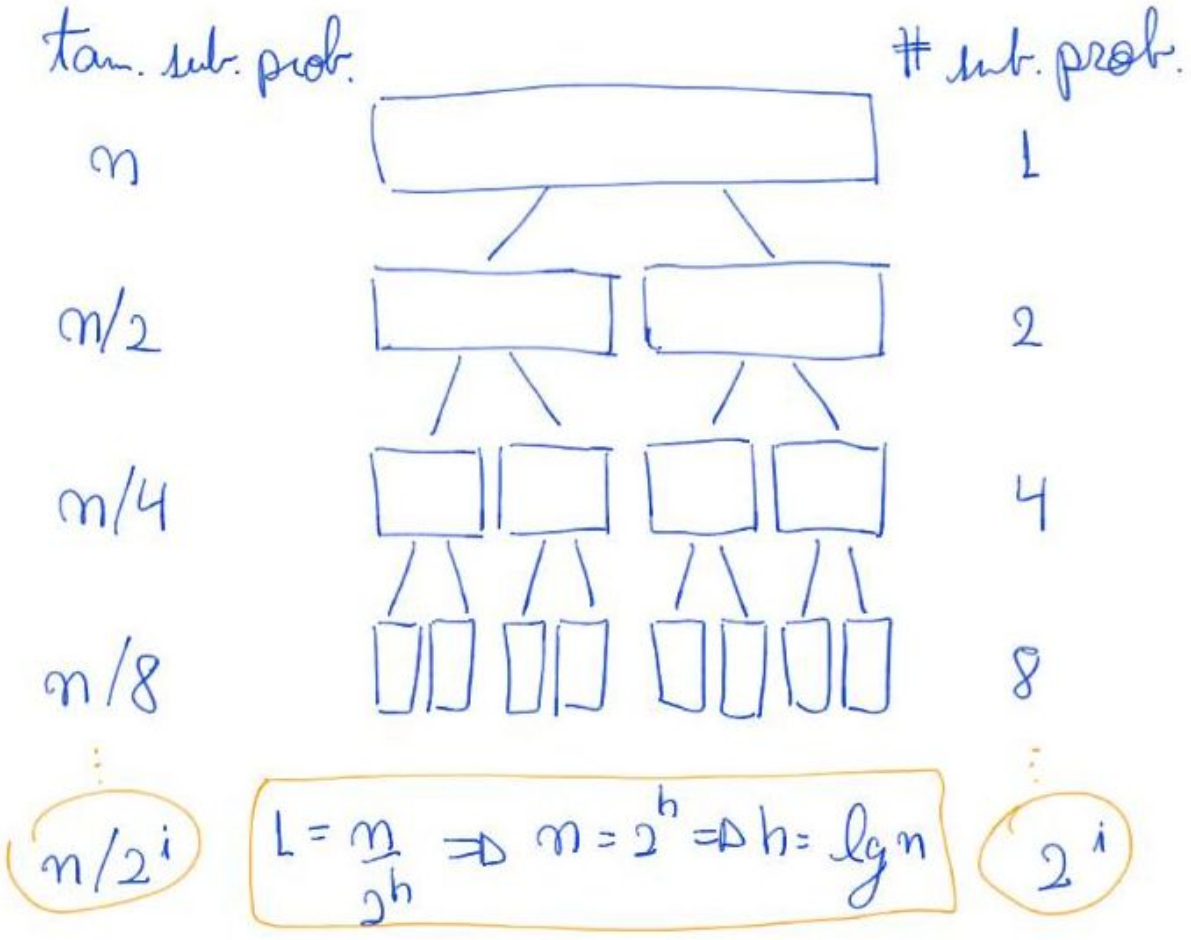
- Você consegue entender por que a função anterior funciona?
- Por que ela não precisa de laços para copiar as sobras do primeiro ou segundo subvetor?
- Quais os invariantes do seu laço principal?
- Ela é estável?

Corretude do mergeSort:

- Pelo caso base p + 1 >= r sabemos que nosso algoritmo devolve
    - subvetores ordenados quando estes tem tamanho menor ou igual a 1.
- Supondo que nosso algoritmo ordena um subvetor de tamanho n/2,
    - verificamos que ele ordena um vetor de tamanho n,
        - uma vez que a função intercala funciona corretamente.
- Note que é necessário provar, usando invariantes, a corretude desta função.

Eficiência de tempo do mergeSort:

- Usamos uma árvore (binária) de recursão na análise.



- O número de níveis da árvore é log_2 n + 1, já que:

    - no nível 0 temos n elementos no vetor,
    - log_2 n é o número de vezes que podemos dividir n por 2
        - antes dele se tornar menor ou igual a 1 (caso base).
- O número de subproblemas no nível j é 2^j.
- O tamanho do vetor dos subproblemas do nível j é n / 2^j.
- Uma chamada do mergeSort realiza basicamente um teste,
    - seguido de duas chamadas recursivas e uma chamada de intercala.
- Como intercala é uma função com eficiência linear,
    - o trabalho não recursivo realizado por mergeSort
        - num vetor de tamanho m é c * m, para alguma constante c.
- Assim, o trabalho realizado por nível da árvore é dado
    - pelo número de subproblemas por nível vezes
        - o trabalho não recursivo realizado por subproblema,
            - i.e., 2^j * c * (n / 2^j) = c*n.
- Por fim, o trabalho total é dado pela soma no número de níveis da árvore
    - do trabalho realizado por nível desta,
        - i.e., \sum_{j=0..log_2 n} c*n = c*n \sum_{j=0..log_2 n} 1
            - = c*n * (1 + log_2 n) = cn log_2 n + cn = O(n log n).
- Numa comparação rápida, para n = 10^6 e 10^9 temos:
    - log_2 n ~= 20 e 30.
    - n log_2 n ~= 2*10^7 e 3*10^10
    - n^2 = 10^12 e 10^18
- Supondo que um computador realize 1 Giga (10^9) operações por segundo:
    - Um algoritmo de ordenação O(n log n) leva, da ordem de,
        - centésimos de segundo e 30 segundos.
    - Um algoritmo de ordenação O(n^2) leva, da ordem de,
        - 16 minutos e 32 anos.

Estabilidade:

- Ordenação é estável.
    - Por que? Mostre que isso vale usando indução.

Eficiência de espaço:

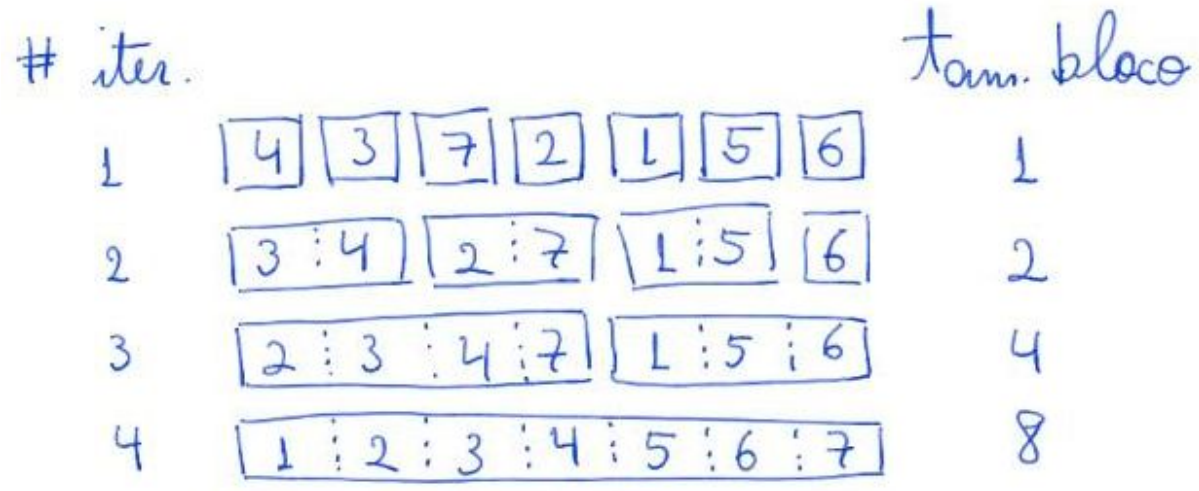
- Ordenação não é in place, pois usa a rotina intercala
    - que precisa de vetor auxiliar (e portanto memória)
        - proporcional ao tamanho dos vetores sendo intercalados.

Curiosidade:

- Podemos usar o algoritmo insertionSort como caso base do mergeSort.
- Isso é interessante porque o insertionSort tem constante menor que o mergeSort, sendo por isso mais rápido quando n é pequeno.

Algoritmo mergeSort iterativo:

- que em cada iteração intercala 2 blocos de tamanho b.




```c
void mergeSortI(int v[], int n)
{
    int b = 1;
    while (b < n)
    {
        int p = 0;
        while (p + b < n)
        {
            int r = p + 2 * b;
            if (r > n)
                r = n;
            intercala1(v, p, p + b, r);
            p = p + 2 * b;
        }
        b = 2 * b;
    }
}
```

Animação:

- Visualization and Comparison of Sorting Algorithms - www.youtube.com/watch?v=ZZuD6iUe3Pc