Programação 2

Classificação Trocas com recursão (QuickSort)

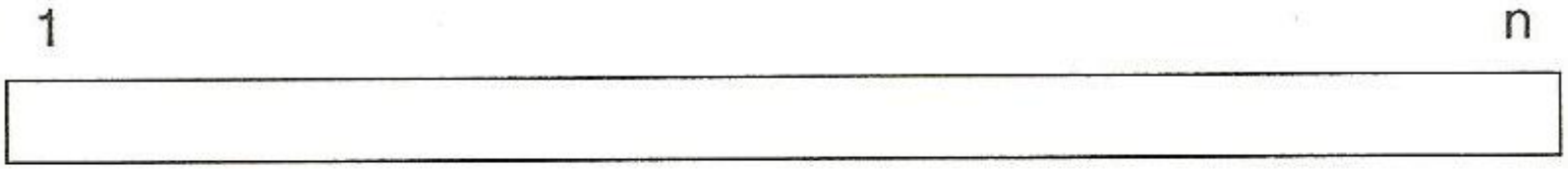
Instituto Federal do Espírito Santo - Ifes
Campus Colatina
Professor: Giovany Frossard Teixeira

QuickSort

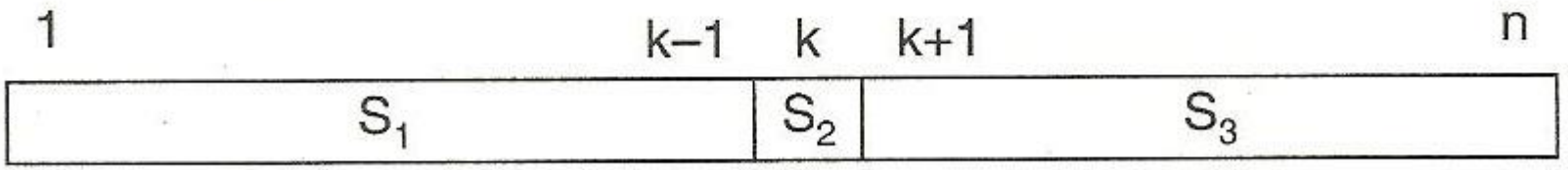
- Seja o vetor de elementos C[1..n] a ser ordenado. Numa etapa inicial, esse vetor é particionado em três segmentos S_1, S_2, S_3, da seguinte forma:
  - S_2 terá comprimento 1 e conterá um elemento denominado particionador;
  - S_1 terá comprimento >= 0 e conterá todos os elementos cujos valores forem menores ou iguais ao do elemento particionador;
  - S_3 terá comprimento >= 0 e conterá todos os elementos cujos valores forem maiores que o elemento particionador.

QuickSort

vetor inicial:



vetor particionado:



QuickSort

```c
int particao(int * vetor, int ini, int fim){
    int pivo = vetor[ini];
    int i = ini + 1;
    int f = fim;

    while (i <= f) {
        if ( vetor[i] < pivo ) {
            i++;
        }
        else if ( pivo < vetor[f] ) {
            f--;
        } else {
            trocar(vetor,i,f);
            i++;
            f--;
        }
    }
    trocar(vetor, ini, f);
    return f;
}
```

```c
// TROCAR dois elementos de posicao
void trocar(int * v, int i, int f){
    int aux;
    aux = v[i];
    v[i] = v[f];
    v[f] = aux;
}
```

QuickSort

```
void quickSort(int * vetor, int ini, int fim){
    int meio;

    if ( ini < fim ) {
        meio = particao(vetor,ini, fim);
        quickSort(vetor,ini,meio-1);
        quickSort(vetor,meio+1,fim);
    }

}
```

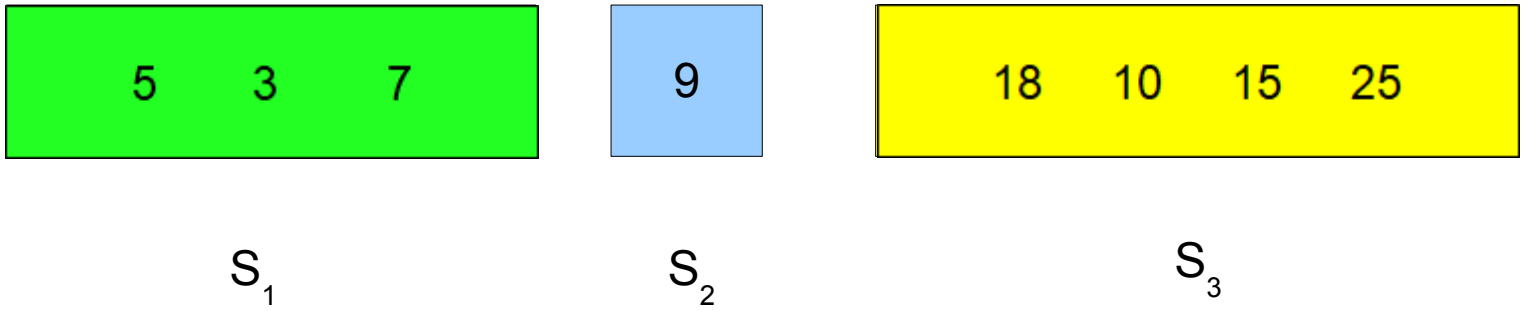
Exemplo

```
void escreveVetor(int *vet, int n){
      for(int i=0; i<n; i++)
         printf("vet[%d] vale %d \n", i, vet[i]);
      printf("\n\n");
}


int _tmain(int argc, _TCHAR* argv[])
{
     int vet[8] = {9, 25, 10, 18, 5, 7, 15, 3};
     quickSort(vet, 0, 7);
     escreveVetor(vet, 8);
     return 0;
}
```

Exemplo

- Quando aplicamos o algoritmo temos, na primeira vez que ***particao*** é chamado o seguinte resultado:



Analise de Desempenho

- A análise que será feita a seguir é baseada na implementação da operação de particionamento (particao) descrita inicialmente.
- Temos a seguinte condição o while:

```
while (i <= f) {
```

- Como i é o inicio e f o fim do vetor, então f-i é igual a n-1, dessa forma, esse método será O(n).

Análise de Desempenho

- Supondo a situação em que o elemento particionador seja tal que o particionamento produza dois segmentos S_1 e S_3 de igual tamanho, teremos dividido o vetor de n elementos em segmentos de acordo com a imagem do slide seguinte:

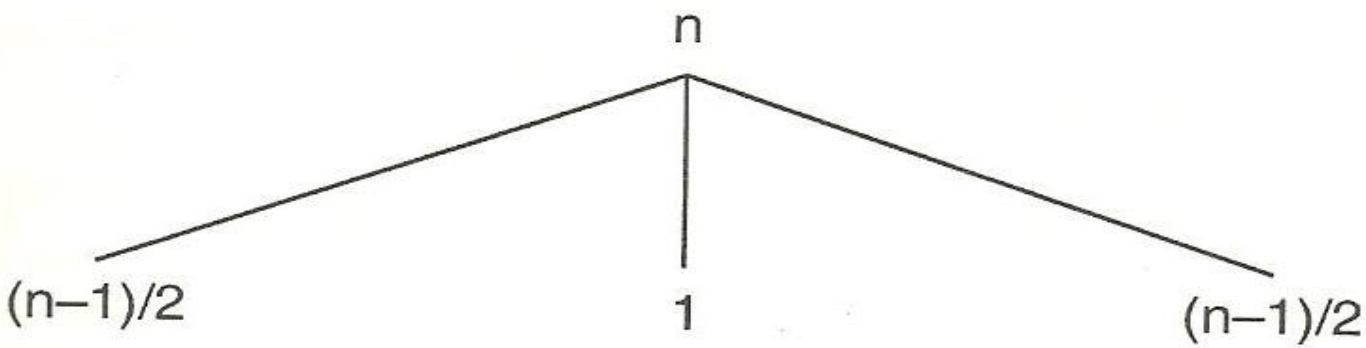
Análise de Desempenho

S_1 com $(n-1)/2$ chaves;

S_2 com 1 chave;

S_3 com $(n-1)/2$ chaves.

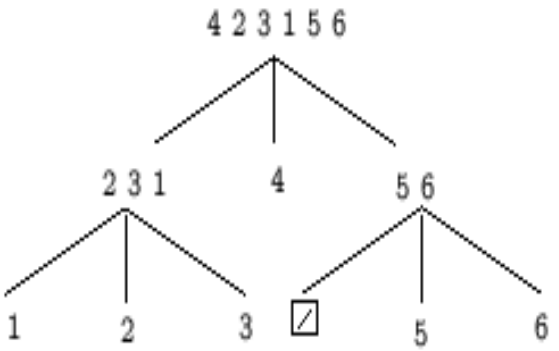
Esquematicamente:

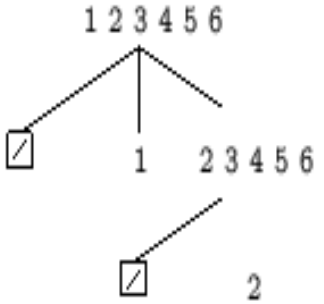


Análise de Desempenho

- Persistindo a situação ideal ao longo dos demais particionamentos teremos a construção de uma árvore de altura log n.
- Assim, melhor caso como cada nível da árvore é O(n) e existem log n níveis, pela regra do produto o algoritmo é O(n * log n).
- O pior caso ocorre quando os particionamentos são de um lado 1 elemento (o particionador) e do outro o número de elementos do nível anterior menos 1.

Análise de Desempenho





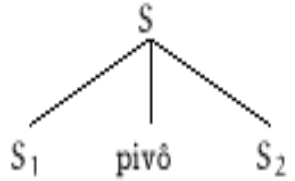
Pior caso

3 4 5 6

4 5 6

5 6

esquema:



Análise de Desempenho

- A altura da árvore no pior caso será n e portanto o algoritmo será $O(n^2)$.
- Segundo Wirth, quando escolhemos uma chave aleatoriamente, o desempenho médio do quicksort se torna inferior ao caso ótimo de um fator da ordem de apenas 2 ln 2, ou seja, uma constante. Assim, para o caso médio, o algoritmo é O(n * log n).

Bibliografia

- AZEREDO, Paulo A. **Métodos de Classificação de Dados**. Rio de Janeiro: Campus,1996.
- SANTOS, Henrique José. **Curso de Linguagem C da UFMG**, apostila.
- FORBELLONE, André Luiz. **Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados.** São Paulo: MAKRON, 1993.