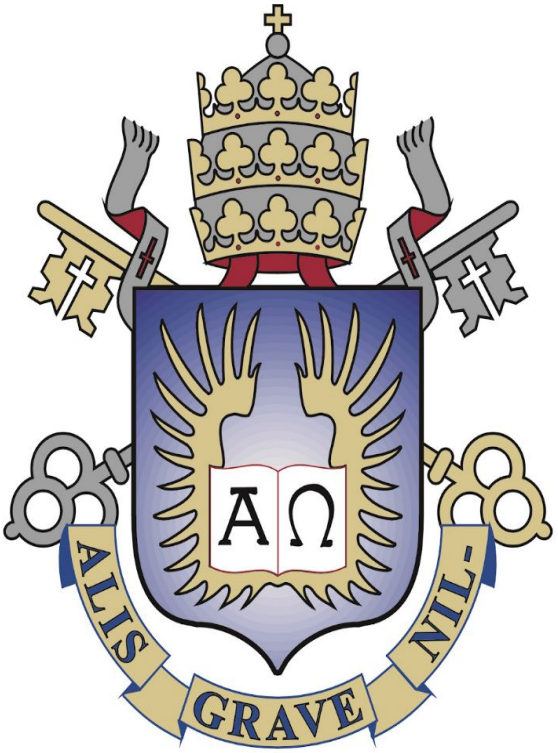
Análise de Algoritmos – Trabalho 1



PUC
RIO

Bianca Fragoso – 1420333

Pedro Felipe Magalhães - 1611074

1. Seleção em tempo linear

1.1 Código da parte principal da LinearSelection:

```
def __linearSelectionRecursion(l,k):
    # Função que vai separando em grupos de 5 até achar a mediana
    m = linearMedian(l)
    # Função que separa L e R
    L,R = getLR(l,m)
    if k == len(L) + 1: # elemento eh a mediana
        return m
    if k < len(L) + 1: # elemento esta a esquerda
        return __linearSelectionRecursion(L,k)
    else: # esta na lista de maiores que a mediana
        #como a lista L e a mediana foram descartadas temos que
compensar no k
        return __linearSelectionRecursion(R , k - len(L) - 1)
```

1.2 Explicação da equação de recorrência

1 - A Função linearMedian() é recursiva. Primeiro ela divide o vetor em grupos de 5 (groupIn5) em O(n).

Código:

```
def linearMedian(l):
    l = groupIn5(l)
    medians = []
    for element in l:
        element.sort()
        medians.append(element[ len(element)//2 ]) #append no elemento
do meio (o ultimo vetor pode ter menos de 5 elem)
    if len(medians) == 1: #achamos a mediana
        return medians[0]

    return linearMedian(medians)
```

Código groupIn5():

```
def groupIn5(l):
    group = []
    for i in range(len(l)):
        if (i % 5) == 0: #devemos criar um novo grupo
            group.append([])
        group[i//5].append(l[i]) # i//5 da o indice

    return group
```

2 - Depois ordena cada grupo, também com uma complexidade de O(n) , visto que, a ordenação de cada grupo é O(5*log(5)) = const, como temos que ordenar n/5 grupos, temos uma complexidade de ordem O(n).

Ela faz a chamada recursiva uma vez, mandando sempre como parâmetro o grupo de medianas, até achar a mediana. No total, ela é O(n).

2 – A função getLR separa o vetor em dois vetores, passando por todo o vetor, se for menor colocando em L e maior colocando em R. Total de O(n).

Código getLR():

```
def getLR(l,mediana):
    L=[]
    R=[]
    # Para tratar o problema de elementos repetidos:
    # se encontrar elementos repetidos, remove o primeiro deles
    # em seguida passa a inserir os elementos repetidos cada hora em uma
das listas
    removed = False
    left = True
    for elemento in l:
        if elemento < mediana:
            L.append(elemento)
        elif elemento > mediana:
            R.append(elemento)
        else:
            if removed == True:
                if(left == True):
                    L.append(elemento)
                    left = False
                else:
                    R.append(elemento)
            else:
                removed = True
    return L,R
```

3 – Depois ela faz as chamadas recursivas.

Portanto, em uma chamada recursiva, A _linearSelectionRecursion tem complexidade O(n).

1.3 Como rodar executável

O script de python deve ser rodado por um interpretador python versão 3+ e tem 3 argumentos possíveis:

--list , para indicar a lista em que vamos procurar o elemento, os números da listas devem ser indicados a seguir separados por espaço Ex:

```
python .\linearSelection.py --list 5 4 6 2 4 1 67 4 4 4 4
```

--k , para indicar a posição do elemento que queremos procurar começando por 1 e indo até n.(o menor elemento da lista tem k = 1)

esse parâmetro é opcional e se não for passado retornaremos o elemento n//2. Ex:

```
python .\linearSelection.py --list 5 4 0 2 4 1 67 4 4 4 4 --k 1
```

retorna o elemento 0

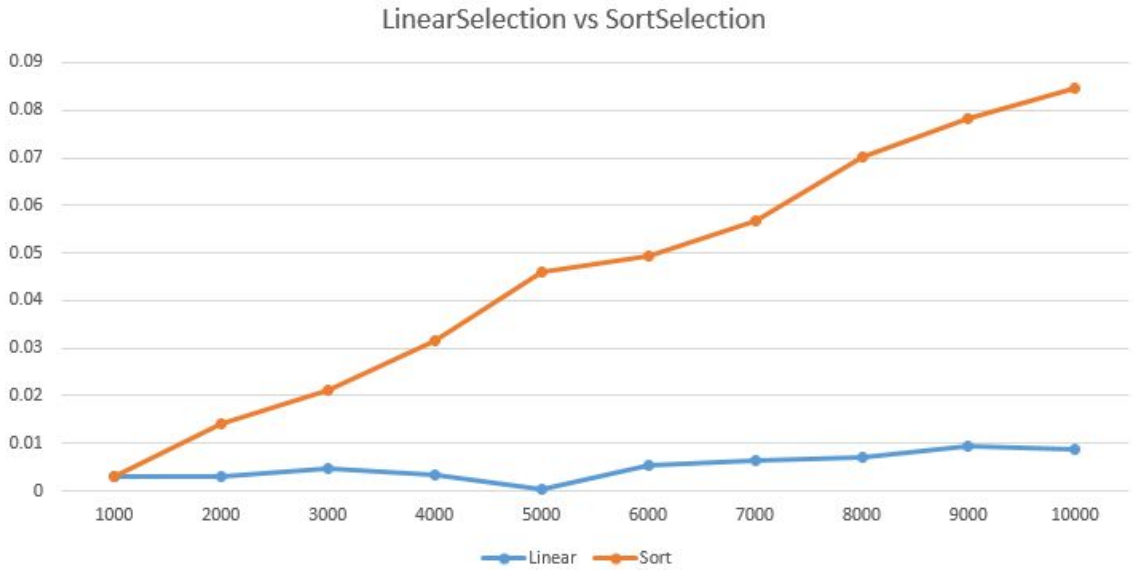
```
python .\linearSelection.py --list 5 4 0 2 4 1 67 4 4 4 4 --k 1
```

retorna o elemento 1

--test , roda os testes propostos imprimindo na tela os tempos e medianas encontradas pelo linear selection e pelo sort selection. Esse parâmetro não pode ser usado junto com o --list, são mutuamente exclusivos

2. Experimentos

2.1 Gráfico



2.2 Explicação dos Gráficos

Os gráficos têm o comportamento esperado, o linear selection tem um crescimento muito mais lento que o sort selection. Isso porque o sort usa um algoritmo de Heapsort que tem complexidade O(nlog(n)) e o linear selection é O(n)