Trabalho 1 - Análise de Algoritmos

Bernardo Vieira Santos - 2220502

Prof.: Marco Molinaro

Tarefa 1: Seleção em tempo linear

1.

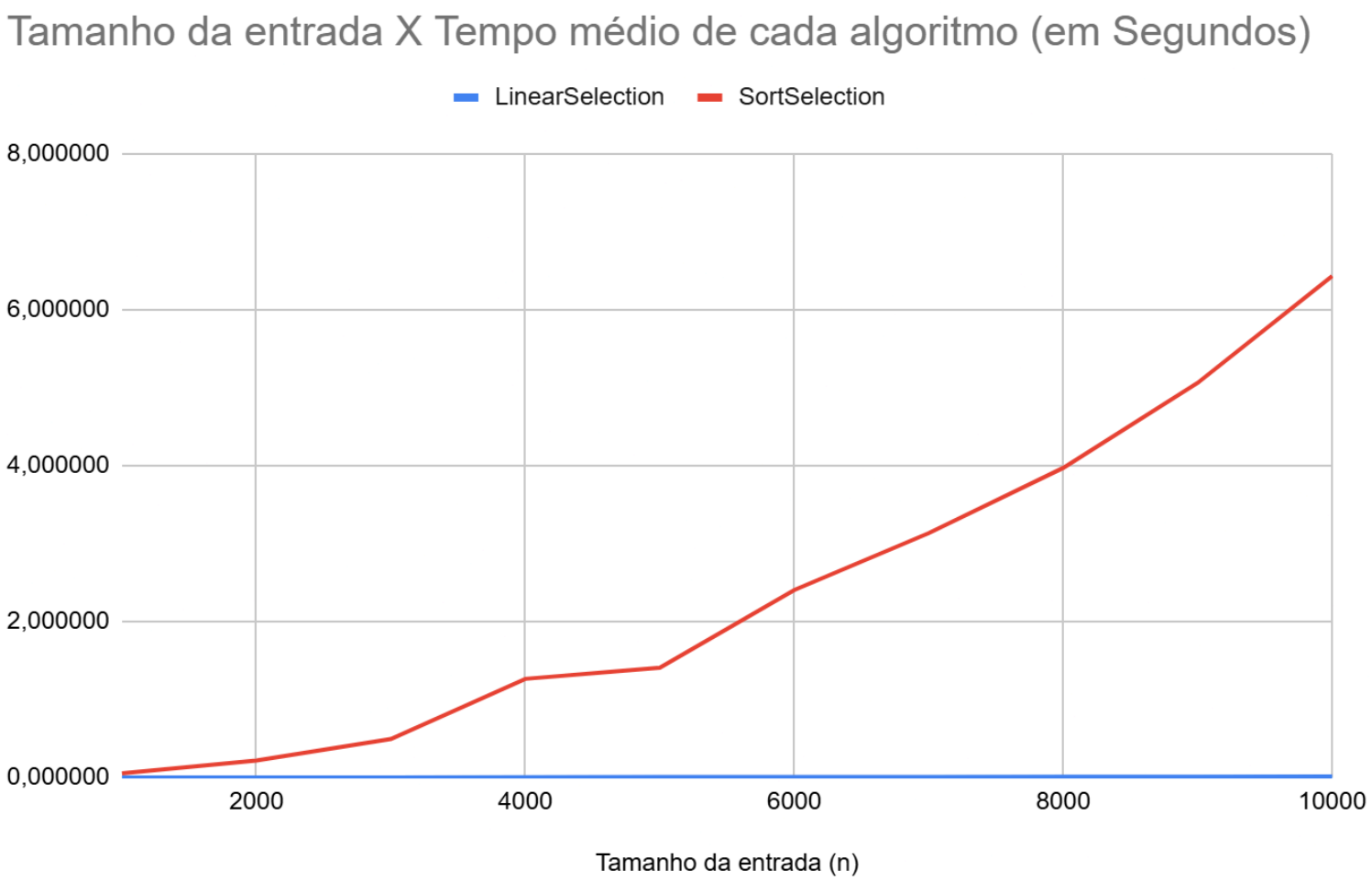
```
groups = size // 5
 if (size % 5) != 0:
   groups += 1

 partitions = []
 list
 for i in range(0, groups):
   heap = Heap(A[i*5:i*5+5])
   partitions.append(heap.heapSort())

medianSet = []
for partition in partitions:
  medianSet.append(partition[len(partition)//2])
  medianSet.sort()

median = linearSelection(medianSet, len(medianSet)//2)

leftPartition = []
rightPartition = []
equalPartition = []
for num in A:
  if num < median:
    leftPartition.append(num)
  elif num > median:
    rightPartition.append(num)
  else:
     equalPartition.append(num)

 lSize = len(leftPartition)
 rSize = len(rightPartition)
 eSize = len(equalPartition)

 if lSize == 0 and rSize == 0:
   return median
```

```
if lSize == k - 1 or (k > lSize and k <= (lSize + eSize)):
  return median
if lSize > k - 1:
  return linearSelection(leftPartition, k)
else:
  return linearSelection(rightPartition, k - lSize - eSize)
```

2.
Como cada partição é um grupo de 5 elementos da lista original A, o uso do HeapSort não afeta a complexidade global, pois cada heap tem tamanho fixo. Mesmo que o HeapSort tenha pior caso O(n log n), aqui n = 5 é constante. Assim, o custo de ordenar depende apenas da quantidade de grupos, ou seja, n. Por isso temos cst * n .

Tarefa 2: Experimentos

1.



2. O primeiro algoritmo implementado apresentou um caráter linear, variando 0.01s entre a menor e a maior entrada, enquanto que o segundo algoritmo cresce de forma mais acentuada, aumentando em mais de 100 vezes entre a menor e a maior entrada, o que é esperado de um perfil quadrático.