MULTITHREADING

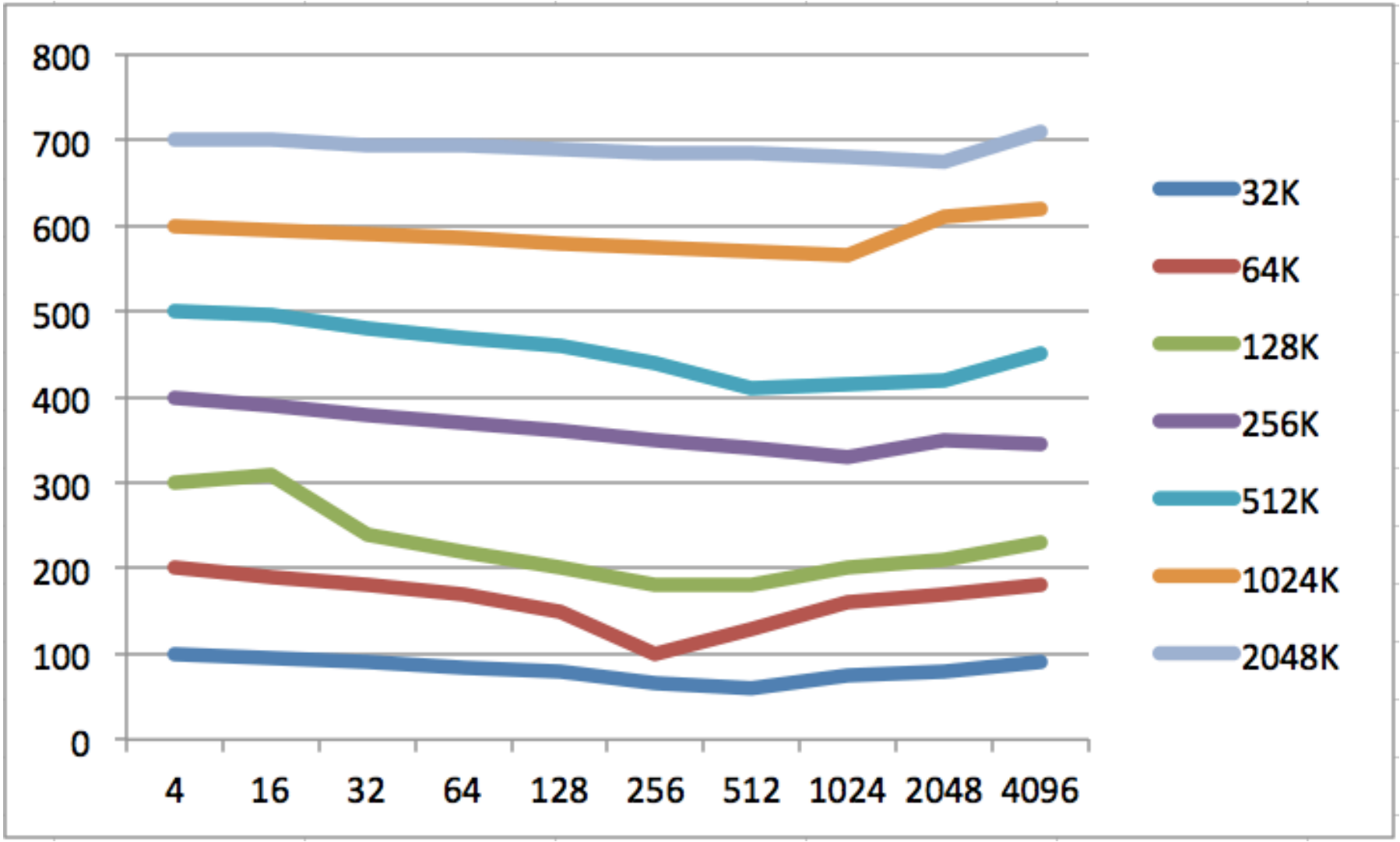
Exercício 1

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
.
.
.
for (i=inicio; i<fim; i++) {
  pthread_mutex_lock(&mutex);
  somatorio += vetor[i];
  pthread_mutex_unlock(&mutex);
}
```

Exercício 1

	32K	64K	128K	256K	512K	1024K	2048K
4	100	200	300	400	500	600	700
16	95	190	310	390	495	595	700
32	90	180	240	380	480	590	695
64	85	170	220	370	470	585	695
128	80	150	200	360	460	580	690
256	65	100	180	350	440	575	685
512	60	130	180	340	410	570	685
1024	75	160	200	330	415	565	680
2048	80	170	210	350	420	610	675
4096	90	180	230	345	450	620	710

Exercício 1


800
700
600
500
400
300
200
100
0
4
16
32
64
128
256
512
1024
2048
4096
32K
64K
128K
256K
512K
1024K
2048K

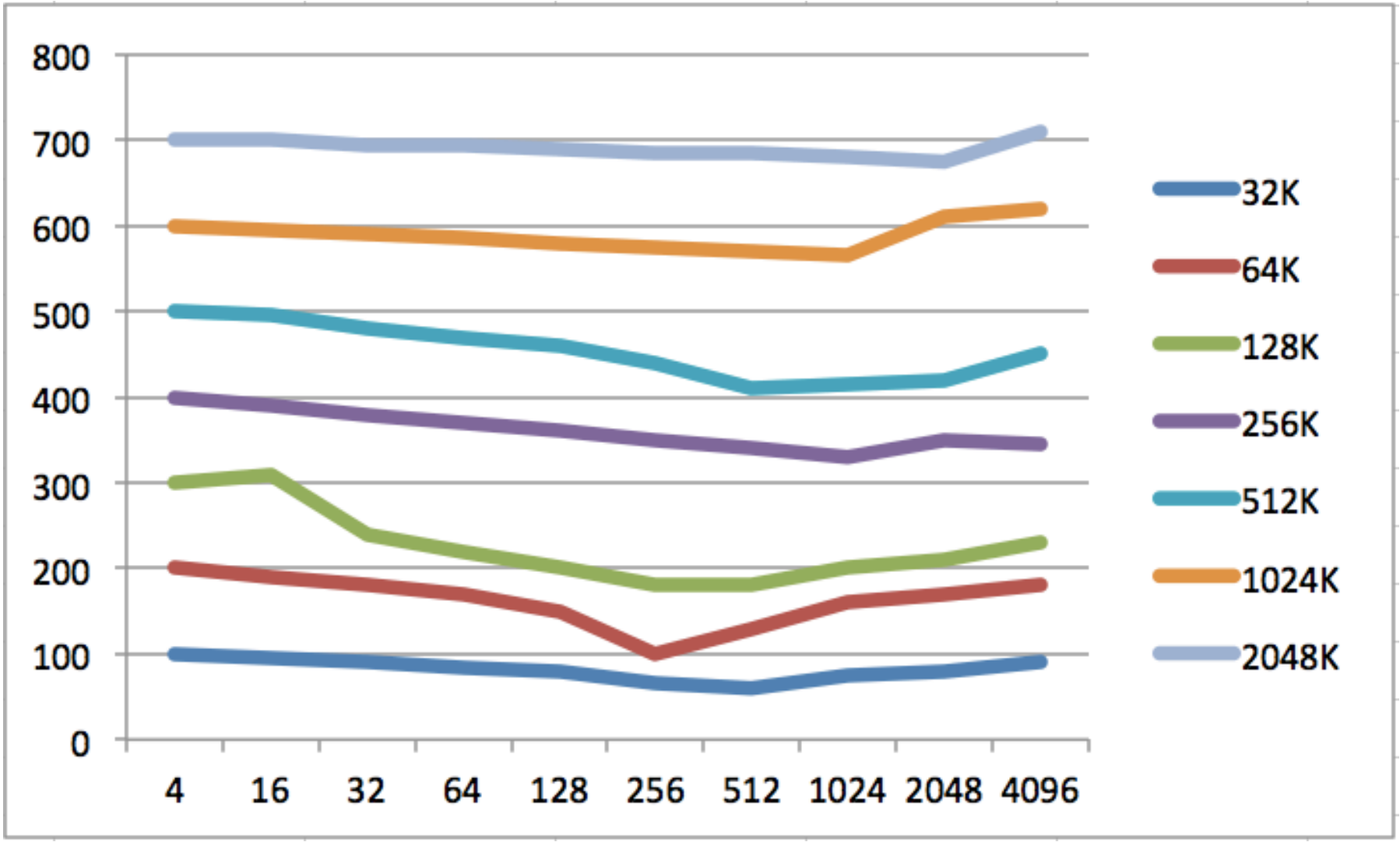
Exercício 2

```
for (i=inicio; i<fim; i++) {
  somatorio[k] += vetor[i];
}
```

Exercício 2

	32K	64K	128K	256K	512K	1024K	2048K
4	100	200	300	400	500	600	700
16	95	190	310	390	495	595	700
32	90	180	240	380	480	590	695
64	85	170	220	370	470	585	695
128	80	150	200	360	460	580	690
256	65	100	180	350	440	575	685
512	60	130	180	340	410	570	685
1024	75	160	200	330	415	565	680
2048	80	170	210	350	420	610	675
4096	90	180	230	345	450	620	710

Exercício 2


800
700
600
500
400
300
200
100
0
4
16
32
64
128
256
512
1024
2048
4096
32K
64K
128K
256K
512K
1024K
2048K

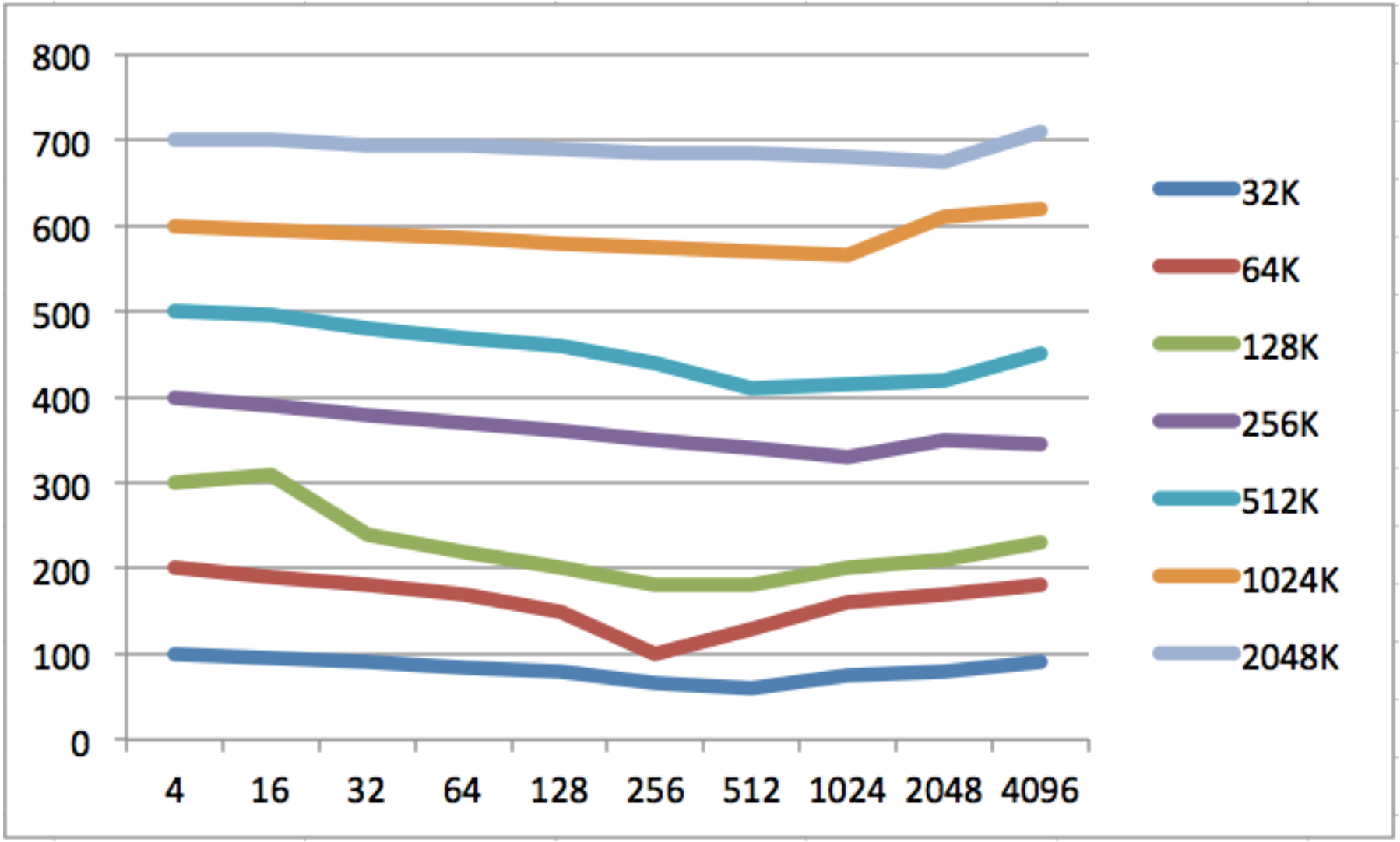
Exercício 3

```
int aux = 0;
for (i=inicio; i<fim; i++) {
   aux += vetor[i];
}
somatorio[k] = aux;
```

Exercício 3

	32K	64K	128K	256K	512K	1024K	2048K
4	100	200	300	400	500	600	700
16	95	190	310	390	495	595	700
32	90	180	240	380	480	590	695
64	85	170	220	370	470	585	695
128	80	150	200	360	460	580	690
256	65	100	180	350	440	575	685
512	60	130	180	340	410	570	685
1024	75	160	200	330	415	565	680
2048	80	170	210	350	420	610	675
4096	90	180	230	345	450	620	710

Exercício 3


800
700
600
500
400
300
200
100
0
4
16
32
64
128
256
512
1024
2048
4096
32K
64K
128K
256K
512K
1024K
2048K

Exercício 4

- Compare os resultados dos três exercícios anteriores
- Faça comentários sobre o tempos de execução quando comparados entre a quantidade de threads e tamanho dos dados
- Inclua nos comentários as situações em que uma estratégia é melhor (ou pior) que as outras

Exercício 5

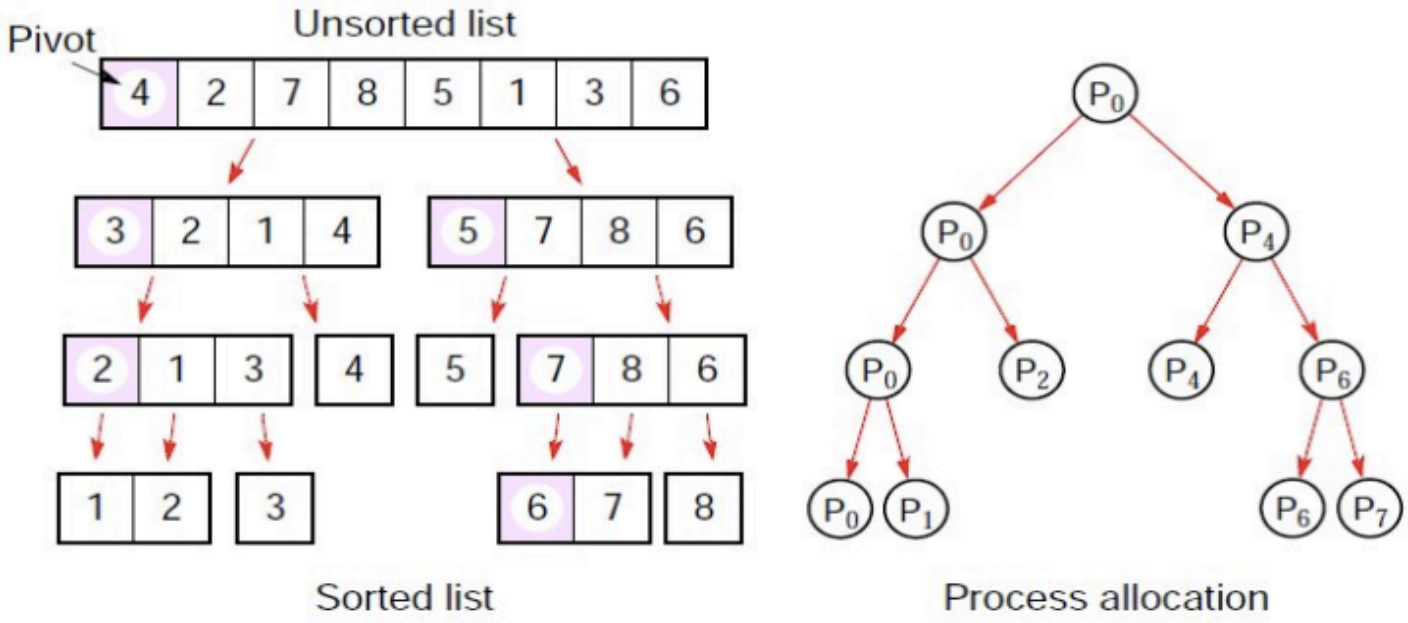
```
procedimento QuickSort(X[], IniVet, FimVet)
var
   i, j, pivo, aux
início
   i <- IniVet
   j <- FimVet
   pivo <- X[(IniVet + FimVet) div 2]
      enquanto(i < j)
        |       enquanto (X[i] <= pivo) faça
        |         |   i <- i + 1
        |       fimEnquanto
        |       enquanto (X[j] > pivo) faça
        |         |   j <- j - 1
        |       fimEnquanto
        |       se (i < j) então
        |         |   aux  <- X[i]
        |         |   X[i] <- X[j]
        |         |   X[j] <- aux
        |       fimSe
        |       i <- i + 1
        |       j <- j - 1
       fimEnquanto
       se (j > IniVet) então
       |  QuickSort(X, IniVet, j)
       fimSe
       se (i < FimVet) então
       |  QuickSort(X, j+1, FimVet)
       fimse
fimprocedimento
```

www.cs.usfca.edu/~galles/visualization/flash.html

Exercício 6

Quicksort em paralelo

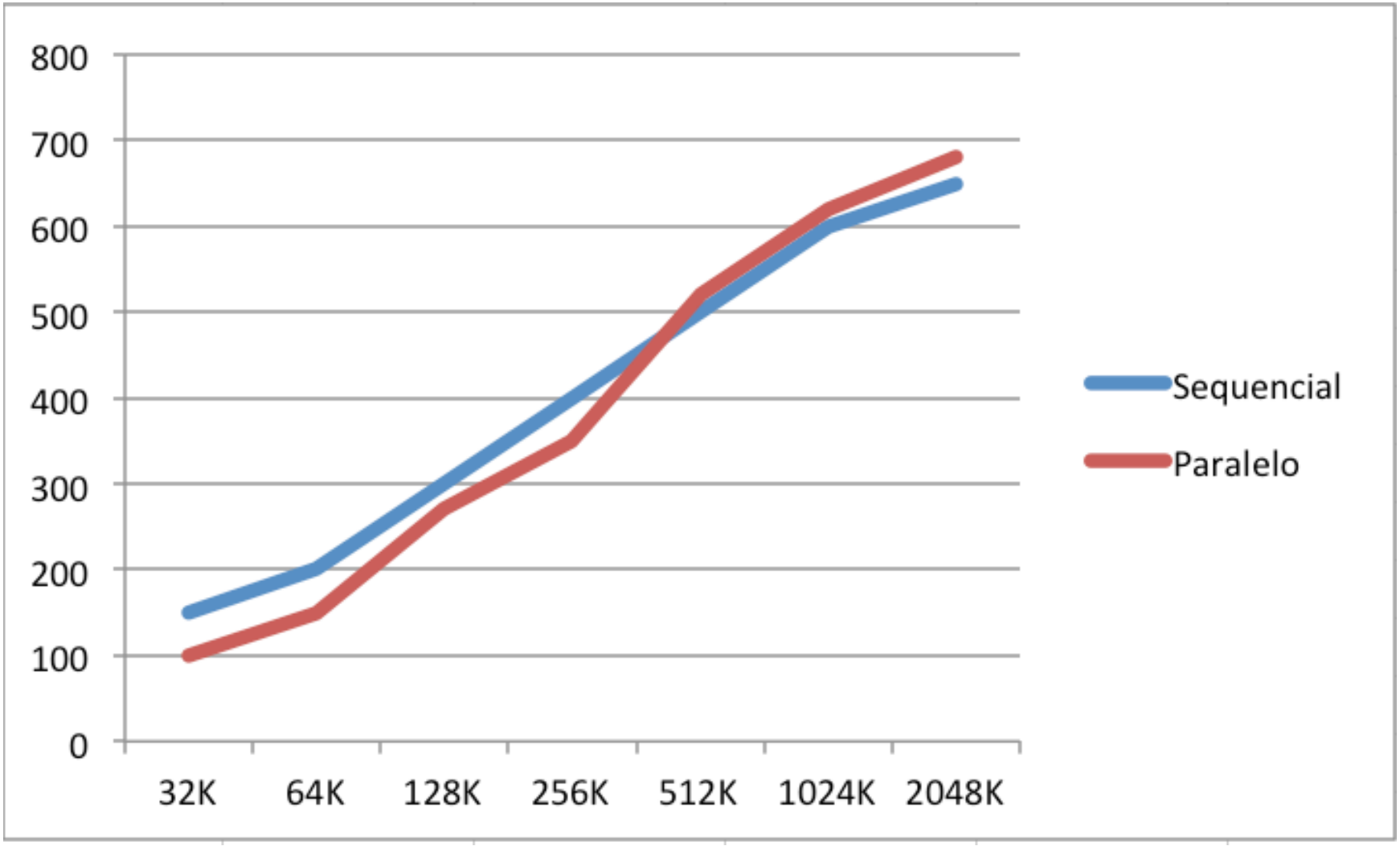
Usando uma atribuição de trabalho a processos em árvore.



Exercício 6

	32K	64K	128K	256K	512K	1024K	2048K
Sequencial	150	200	300	400	500	600	650
Paralelo	100	150	270	350	460	580	620

Exercício 6


800
700
600
500
400
300
200
100
0
32K
64K
128K
256K
512K
1024K
2048K
Sequencial
Paralelo

Exercício 6

- Adicione no relatório uma **comparação** entre os resultados do Quicksort sequencial com o do paralelo, simplesmente justificando as razões para os resultados obtidos para cada quantidade de dados (N)