UNIVERSIDADE FEDERAL DO AMAZONAS

Trabalho Prático 1

Projeto e Análise de Algoritmos

Diogo Soares Moreira

2120184

Manaus, Am. 2012

Questão 1:

a) $\sum_{i=a}^{n} k^i \rightarrow$

$Sn = \sum_{i=a}^{n} k^i$

$Sn = k^a + k^{(a+1)} + \cdots + k^n \quad (1)$

$k * Sn = k^{(a+1)} + k^{(a+2)} + \cdots + k^n + k^{(n+1)} \quad (2)$

Somando os termos (1) *e* (2) *teremos o corte vertical de todos os elementos menos o primeiro elemento de* (1) *e o último de* (2). *Logo*:

$-Sn + k * Sn = k^a - k^{(n+1)}$

$Sn = \frac{k^a - k^{(n+1)}}{-1+k}$

b) $\sum_{i=1}^{n} \log i \rightarrow$

$\log 1 + \log 2 + \ldots + \log n \rightarrow \log\left(\prod_{1}^{n} i\right) = \log n!$

que pela aproximação de Stirling é: $\log(n!) = \Theta(n \log n)$,
por propriedade de Stirling.

c) $\sum_{1}^{(n-1)} \frac{1}{i(i+1)} \rightarrow \frac{1}{i(i+1)} = \frac{1}{i} - \frac{1}{i+1} \rightarrow \sum_{1}^{(n-1)} \left(\frac{1}{i} - \frac{1}{i+1}\right) = \sum_{1}^{(n-1)} \frac{1}{i} - \sum_{1}^{(n-1)} \frac{1}{i+1} \rightarrow$

$\left(\frac{1}{1} - \frac{1}{2}\right) + \left(\frac{1}{2} - \frac{1}{3}\right) + \left(\frac{1}{3} - \frac{1}{4}\right) + \cdots + \left(\frac{1}{n} - \frac{1}{n+1}\right) \rightarrow$

Tal soma é então convertida para :

$1 - \frac{1}{n+1} \rightarrow$

Aplicando o limite sobre a função encontrada teremos a convergência abaixo :

$\lim_{n \to \infty} 1 - \frac{1}{n+1} = 1$

d) $\sum_{0}^{n}(b^{(2i+1)}-bi)\rightarrow$

$\sum_{0}^{n}b^{(2i+1)}-\sum_{0}^{n}bi=\sum_{0}^{n}b^{(2i+1)}-b\sum_{0}^{n}bi$

$(1)\sum_{0}^{n}i=\frac{[(n+1)n]}{2}$

$\sum_{0}^{n}b^{(2i+1)}=b\sum_{0}^{n}b^{2i}=Sn$

$-Sn+b^{2}Sn=-(1+b^{2}+b^{4}+..+b^{2n})+(b^{2}+b^{4}+..+b^{(2n+2)})\rightarrow$

$(b^{2}-1)Sn=-1+b^{(2n+2)}\rightarrow Sn=\frac{(-1+b^{(2n+2)})}{b^{2}-1}(2)$

Juntando os passos $(1)e(2)$, *teremos agora o resultado*:

$\frac{(-1+b^{(2n+2)})}{b^{2}-1}-b\frac{[(n+1)n]}{2}$

e)

Questão 2:

a) $n\log_{4}n+n=\Theta(n\log n)\rightarrow$

Então a prova parte da definição

$c1(n\log n)\leq n\log_{4}n+n\leq c2(n\log n)\rightarrow$

Dividindo toda a expressão por n:

$c1(\log n)\leq\log_{4}n+1\leq c2(\log n)\rightarrow$

$\log_{4}n=\frac{\log n}{\log 4}$, *por propriedade logarítimica*

$c1(\log n)\leq\frac{\log n}{\log 4}+1\leq c2(\log n)$

Dividingo toda a expressão por $\log n$, *teremos*:

$c1\leq\frac{1}{2}+\frac{1}{\log n}\leq c2$

Para $c1=\frac{1}{2}$ $e\ c2\geq 1$. *Logo a afirmação é verdadeira.*

b) $2^{(n+3)}=O(2^{n})\rightarrow$

Verdadeiro se $\rightarrow\exists c\wedge n_{0}\rightarrow 2^{(n+3)}\leq c2^{n}\ \forall\ n\geq n_{0}$

$2^{n}(2^{3})\leq c2^{n}\rightarrow$

Dividindo todos os lados por 2^{n}:

$2^{3}\leq c$

$8\leq c$, *logo para para* $c\geq 8$, *a afirmação inicial é verdade.*

c) $n^b = \omega(n^a) \cdots 0 < a < b \rightarrow$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \rightarrow$

$\lim_{n\to\infty} \frac{n^b}{n^a} = \infty \rightarrow$

como $b > a$ *, logo* $n^b > n^a$ *, portanto o mesmo tem o crescimento assintótico muito maior levando o limite abaixo a ser verdadeiro.*

$\lim_{n\to\infty} \frac{n^{(b-a)} n^a}{n^a} = \infty \rightarrow$

$\lim_{n\to\infty} n^{(b-a)} = \infty$ *, tal que* $b - a > 0$

d) $n \log n = o(n^3) \rightarrow$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \rightarrow \lim_{n\to\infty} \frac{n \log n}{n^3} = 0 \rightarrow \lim_{n\to\infty} \frac{\log n}{n^2} = 0 \rightarrow$

Como n^2 *cresce numéricamente e* $\log n$ *com* $n \to \infty$ *cresce muito menos que* n^2*, logo é verdadeiro afirmar* :

$\lim_{n\to\infty} \frac{\log n}{n^2} = 0$

Portanto : $n \log n = o(n^3)$.

e) $10^{-8} n^2 \neq o(n^2) \rightarrow$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \rightarrow \lim_{n\to\infty} \frac{10^{-8} n^2}{n^2} = 0 \rightarrow$

$\lim_{n\to\infty} 10^{-8} = 0$ *(Dividindo os* n^2*)*

Contudo limite de 10^{-8} *é o mesmo que* $\lim k$ *, com k constante ,*

logo $\lim_{n\to\infty} 10^{-8} = 10^{-8}$*, o que contraria a definição e não satisfaz uma igualdade em que* $10^{-8} n^2 = o(n^2)$ *, portanto elas são diferentes.*

Questão 3:

a)

$$T(n)=2\,T\left(\frac{n}{2}\right)+(n-1)$$

Usando o método da iteração:

$$\rightarrow 2\,T\left(\frac{n}{2}\right)+(n-1)$$

$$\rightarrow 2(2\,T\left(\frac{n}{2^2}\right)+\frac{n}{2}-1)+(n-1)$$

$$\rightarrow 2^2\,T\left(\frac{n}{2^2}\right)+2\mathrm{n}-3$$

$$\rightarrow 2^2(2\,T\left(\frac{n}{2^3}\right)+\frac{n}{2^2}-1)+2\mathrm{n}-3$$

$$\rightarrow 2^3\,T\left(\frac{n}{2^3}\right)+3\mathrm{n}-7$$

$$\rightarrow 2^4(2\,T\left(\frac{n}{2^4}\right)+\frac{n}{2^3}-1)+3\mathrm{n}-7$$

$$\rightarrow 2^4\,T\left(\frac{n}{2^4}\right)+4\mathrm{n}-15$$

$$\vdots$$

$$\rightarrow 2^k\,T\left(\frac{n}{2^k}\right)+kn-(2^k-1)$$

Tomando $n=2^k \rightarrow k=\log n$ *e substituindo as variáveis*

$$\rightarrow n\,T(1)+n\log n-(n-1)$$

$$\rightarrow n+n\log n-n+1$$

$$\rightarrow n\log n+1$$

Enfim temos que:

$$\rightarrow T(n)=n\log n+1$$

b) $T(n)=2\,T(\frac{n}{4})+3\mathrm{n}$

Usando iterative method novamente:

$$2\left(2\,T\left(\frac{n}{4^2}\right)+\frac{3\mathrm{n}}{4}\right)+3\mathrm{n}$$

$$2^2 T\left(\frac{n}{4^2}\right)+3\mathrm{n}\frac{3}{2}$$

$$2^3 T\left(\frac{n}{4^3}\right)+3\mathrm{n}\frac{7}{2^2}$$

$$\vdots$$

$$2^k T\left(\frac{n}{4^k}\right)+3\mathrm{n}\frac{2^k-1}{2^{k-1}}$$

$$2^{2^k}=4^k=n$$

$$k=\log_4 n\rightarrow 2^k=\sqrt{n}$$

Fazendo as devidas substituições:

$$T(n)=\sqrt{n}+3\mathrm{n}\frac{\sqrt{r}-1}{\sqrt{n}\,2^{-1}}$$

$$T(n)=\sqrt{n}+\frac{3\mathrm{n}}{2^{-1}}-\frac{3\mathrm{n}}{\sqrt{n}\,2^{-1}}$$

c) $T\left(\frac{2\mathrm{n}}{3}\right)+1\rightarrow$

Usando o método iterativo podemos obter:

$$T(n)=T\left(\frac{2\mathrm{n}}{3}\right)+1$$

$$T\left(\frac{2\mathrm{n}}{3}\right)=T\left(2^2\frac{n}{3^2}\right)+1$$

$$T\left(2^2\frac{n}{3^2}\right)=T\left(2^3\frac{n}{3^3}\right)+1$$

$$\vdots$$

$$T\left(2^{k-1}\frac{n}{3^{k-1}}\right)=T\left(2^k\frac{n}{3^k}\right)+1$$

Cortando as transições chegaremos em:

$$T(n)=T\left(2^k\frac{n}{3^k}\right)+\sum_{n\rightarrow 1}^{k}1\rightarrow \textit{Usando } k=\log_{\frac{2}{3}}\frac{1}{n}\textit{ teremos}:$$

$$T(n)=\log_{\frac{2}{3}}\frac{1}{n}+1$$

d)

$T(n)=T(\sqrt{n})+1, T(2)=1$

Usando o método iterativo:

$T(n)=T(\sqrt{n})+1$

$T(n^{\frac{1}{2}})=T(n^{\frac{1}{2^2}})+1$

$T(n^{\frac{1}{2^2}})=T(n^{\frac{1}{2^4}})+1$

$\vdots$

$T(n^{\frac{1}{2^k}})+1$

Cortando as diferenças teremos:

Assumindo $n=2^{2^k}$ *e* $k=\log(\log n)$

$(k+1)+1$

$\log(\log n)+1+1 \rightarrow \log(\log n)+2$

Questão 4:

a)

$T(n)=8T\left(\frac{n}{2}\right)+n^2+(n!)^{-2}$

Primeiro caso: $f(n)=O(n^{\log_b a-\epsilon})$:

Como $\frac{1}{n!^{-2}}$ *tende a* 0, *consideramos* $f(n)$ *como* n^2 *em complexidade.*

$\log_b a \rightarrow \log_2 8=3$

$f(n)=O(n^{\log_b a-\epsilon})\rightarrow$

$n^2=O(n^{3-\epsilon})$

Para $\epsilon=1$, *logo a condição é verdadeira pois* $n^2=O(n^2)$.

Portanto $T(n)=\theta(n^{\log_b a})\rightarrow T(n)=\theta(n^3)$

b)

$T(n)=4T\left(\frac{n}{4}\right)+2\mathrm{n}$

Usando o segundo caso: $f(n)=\theta(n^{\log_b a})$:

Considerando $f(n)$ *como* $2\mathrm{n}$ *em complexidade* n:

$\log_b a \rightarrow \log_4 4=1$

$f(n)=\theta(n^{\log_b a})\rightarrow$

$2\mathrm{n}=\theta(n^1)$

Logo a afirmação é verdadeira.

Portanto $T(n)=\theta(n^{\log_b a}\log n)$

$T(n)=\theta(n\log n)$

c) $T(n)=2T\left(\frac{n}{3}\right)+k^n$, *tal que* $0<k\leq 1$

Usando a função, podemos compara−la da seguinte forma:

$f\frac{(n)}{n^{\log_b a}}=\frac{n^k}{n_3^{\log}2}$

$\frac{n^k}{n_3^{\log}2}<n^{\epsilon}$ *para qualque* $\epsilon>0$

Logo as diferenças entre as funções não são polinomias, portanto o teorema mestre não pode ser aplicado

d) $T(n)=2T\left(\frac{n}{3}\right)+k^n$, *tal que* $k>1$

Usando o terceiro caso do teorema mestre:

$f(n)=\Omega(n^{\log_3 2}+\epsilon)$

Então provamos a condição de regularidade $af(\frac{n}{b})\leq cf(n)$.

$2\mathrm{k}^{\frac{n}{3}}\leq ck^n$

Para $c<1$, *podemos utilizar* $c=\frac{1}{2}$ *e então*

a sentenca será verdadeira para n muito largo.

Com a condição satisfeita, então temos:

$T(n)=\theta(f(n))\rightarrow T(n)=\theta(k^n)$

Questão 5:

a)

b) *Fórmula que encontrei*:

$$T(n)=T\left(\frac{n}{2}\right)+n$$

Expandindo:

$$T(n)=T\left(\frac{n}{3}\right)+n$$

$$T\left(\frac{n}{3^2}\right)+\frac{n}{3}$$

$$T\left(\frac{n}{3^3}\right)+\frac{n}{3^2}$$

$$\vdots$$

$$T\left(\frac{n}{3^k}\right)+\frac{n}{3^k}-1$$

Fazendo as devidas somas:

$$T(n)=T\left(\frac{n}{2^k}\right)+n\sum_{0}^{k-1}\frac{1}{3^i}$$

$$\sum_{0}^{k-1}\frac{1}{3^i}=Sn$$

$$-Sn+\frac{1}{3}Sn=1+\frac{1}{3^k}$$

$$Sn=\frac{1+\frac{1}{3^k}}{-1+\frac{1}{3}}$$

Fazendo $3^k=n\rightarrow k=\log_3 n$

$$T(n)=1+\frac{1+\frac{1}{\log_3 n}}{-1+\frac{1}{3}}$$

Logo a complexidade é da ordem de $O(\log_3 n)$

c) *Para esta questão precisei encontrar os laços , ou melhor o custo dos laços (Considerando o pior caso neste momento)*
Considerando os custos de if + de troca como constantes , ie , $O(1)$, então :
Para o primeiro laço , temos um custo linear para ele , logo $O(n)$
E para o segundo laço temos que ele entra no máximo $\frac{n-1}{2}$ vezes

$T(n)=\frac{n(n-1)}{2}+O(1)$, *logo a complexidade para o pior caso é* $O(n^2)$.

Caso médio

No pior caso teremos : $T(n)=\frac{n(n-1)}{4}+O(1)$, *mantendo a complexidade em* $\theta(n^2)$

Melhor caso
O melhor caso seria quando o vetor estaria ordenada e todas as condições de loop falhariam.
Então apenas o primeiro loop seria percorrido , logo :
$T(n)=n+O(1)\rightarrow$ *resultando em complexidade linear.*

Questão 6:

Essa questão desenvolvi em C/C++ o código, abaixo está o algoritmo principal do código e o que fazia as permutações pedidas na questão.

```
void algoll2(int n1, int a, int b){
    qsort(vet,n1,sizeof(int),compare);      // Complexidade = Θ(n log n)
    int i = 0;
    while(vet[i]<=a && i<n1){       //Complexidade Linear para varrer todo o vetor
        p1.push_back(vet[i]);       //Separado em 3 whiles que correm juntos
        i++;                        //em Θ(n)
    }
    while(vet[i]<=b && i<n1){
        p2.push_back(vet[i]);
        i++;
    }
    while(i<n1){
        p3.push_back(vet[i]);
        i++;
```

```
        }                                   //ordena novamente com quicksort
        sort(p1.rbegin(), p1.rend());       //Complexidade alcançada Θ(n log n)
        sort(p3.rbegin(), p3.rend());
}
```

Logo haverá T(n) = 3(n log n) + n

T(n) → Θ(n log n), pois n log n > n. Essa foi a melhor solução que pude encontrar em ordem de complexidade.

Questão 7:

Relatório de Análise do Desempenho de Algoritmos de Ordenação

Introdução

Esta questão/relatório tinha como objetivo a implementação de 3 algoritmos de ordenação clássicos e analisa-los com um conjunto de entradas afim de valida-los em termos de pesquisa o seu uso e custo operacional. Os três algoritmos implementados eu descrevo abaixo a ideia básica e minha idéia de implementação.

1. Heapsort → Algoritmo que possui uma ordenação básica própria que parte das premissas da sua definição, onde um elemento seja ele o maior ou menor fica no topo enquanto que cada filho seu é menor que ele ou maior, dependendo do tipo de implementação do heap. A minha ideia de implementação foi criar um laço de ordenação que para cada elemento central do vetor, busco o seu filho e realizo as trocas no array até que o filho seja menor que o tamanho máximo do vetor. O laço principal é então realizado até que o tenhamos alcançada o tamanho mínimo de quebra do vetor

2. Mergesort → Algoritmo clássico no qual me basiei simplesmente na ideia passada pelo livro texto do cormen. No qual dividi-se recursivamente o vetor de dois em dois, jogando a intercalação em um vetor temporário em cada tamanho pequeno de vetor, até que se chegue ao tamanho máximo do vetor através da sequência finalizada de recursões.

3. Quicksort → Outro algoritmo clássico que assume um caso médio semelhante ao pior caso do Mergesort, com direito até a biblioteca própria nativa em muitas linguagens como o c++/c. A minha estratégia de implementação dele foi 100% baseado na do livro texto do cormen. Devido a sua engenhosidade fácil e entendível com clareza, não busquei na literatura outras maneiras de implementação, usando apenas o conceito básico de particionamento do vetor em duas estruturas com elementos maiores que ele e com menores em outra estrutura e ordenando-os a partir dessa premissa.

Outro passo chave para a realização deste relatório foram as decisões de projeto. A primeira decisão de projeto escolhida foi utilizar potências na base, mais especificamente, arrays de mil, dez mil, cem mil e hum milhão de elementos. Todavia, um pequeno problema de alocação de memória no caso de teste de hum milhão para elementos do tipo integer sendo eles 0's ou 1's fez com que uma nova decisão de projeto fosse tomada. Então foi assumido em outro momento no meu trabalho que os arrays seriam de tamanhos incrementantes de 50 em 50. Logo foi escolhido, afim de manter uma base boa de elevação e também evitar possíveis erros de falha de segmentação na linguagem os seguintes tamanho: 10 mil, 50 mil, 100 mil e 150 mil.

Para cada extensão de tamanho de estruturas (50 mil, 100 mil, etc) é usado apenas um gerador de números aleatórios de trivial implementação. Os vetores gerados são os mesmos

usados em todos os 3 algoritmos afim de evitar conferir-se vantagem sobre melhores casos ou piores casos de um algoritmo ou outro.

Quanto a coleta temporal de execução do algoritmo em cada momento de execução foi usado a biblioteca do C time.h. Então foi usado método clock() para se buscar o tempo de execução em segundos da execução de cada algoritmo para cada estrutura de entrada.

A análise dos resultados foram feitas em apenas um computador com configuração core 2 duo de 2.6 Ghz. E a análise estatística e gráfica dos resultados foi feita utilizando a ferramenta Minitab para análise estatística e operacional de dados.

Caso 1: Short Int

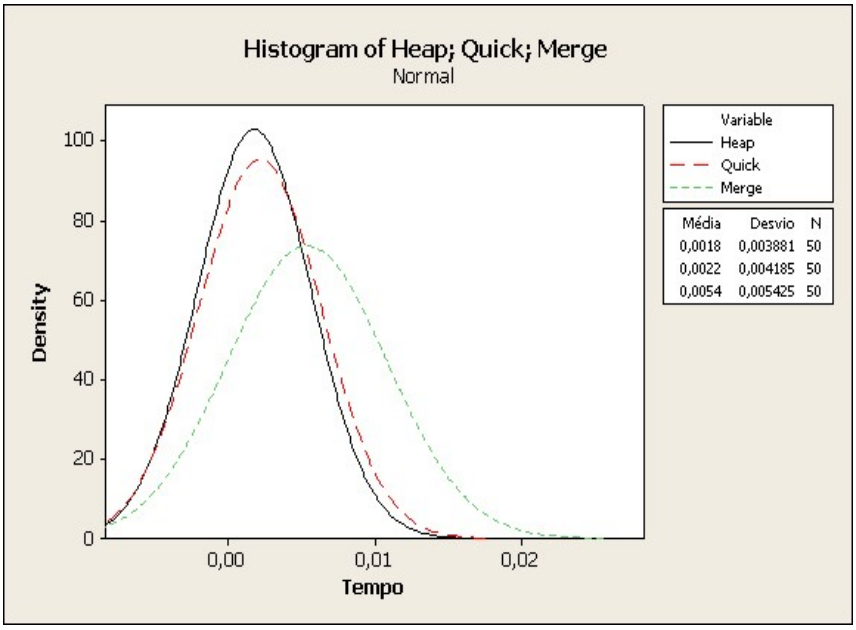
O primeiro caso que viso avaliar neste momento é o caso que considei como base para testes, isto é, o caso onde a estrutura de entrada só possui inteiros entre 0 e 65535 ou em C, o tipo unsigned short int.

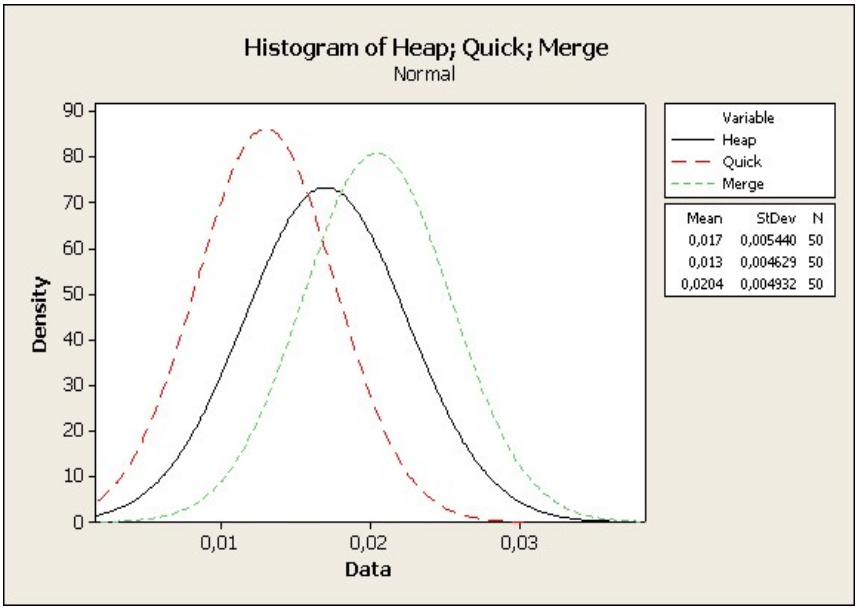
A tabela com as médias de tempo e o desvio padrão (com 4 casas de precisão), respectivamente são apresentadas para cada caso são apresentadas abaixo:

	Heapsort		Mergesort		Quicksort	
Caso 1 (10 Mil)	0.0017	0.0384	0.0054	0.0053	0.0021	0.0041
Caso 2 (50 Mil)	0.0170	0.0053	0.0204	0.0048	0.0130	0.0045
Caso 3 (100 Mil)	0.0358	0.0056	0.0432	0.0054	0.0282	0.0038
Caso 4 (150 Mil)	0.0572	0.0060	0.0681	0.0047	0.0438	0.0052

Tabela 1. Média e desvio padrão para os casos de teste com short int.

Logo abaixo, coloco uma sequência de gráficos onde é possível ver uma leve padronização para o desempenho destes algoritmos para os casos acima.


Histogram of Heap; Quick; Merge
Normal
Density
Tempo
0,00
0,01
0,02
0
20
40
60
80
100
Variable
Heap
Quick
Merge
Média Desvio N
0,0018 0,003881 50
0,0022 0,004185 50
0,0054 0,005425 50


Histogram of Heap; Quick; Merge
Normal
Density
Data
0,01
0,02
0,03
0
10
20
30
40
50
60
70
80
90
Variable
Heap
Quick
Merge
Mean StDev N
0,017 0,005440 50
0,013 0,004629 50
0,0204 0,004932 50

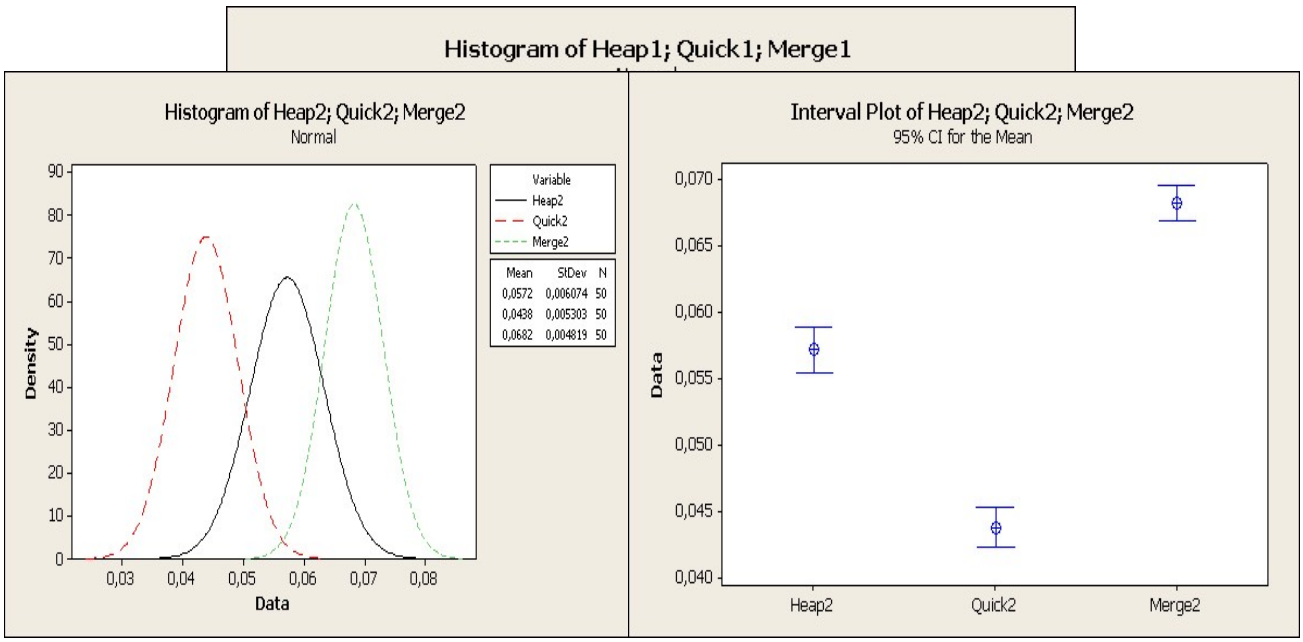


Gráfico 1. Sequência linear de grafos que demonstra o acompanhamento linear do Quicksort frente aos seus componentes ditos aqui de rivais.

Podemos notar pelos gráficos e pelo intervalo de confiança da figura acima da direta que enquanto o merge sempre tendeu a ficar a frente em tempo de execução para os casos avaliados, o heap o acompanhou, enquanto que aparenta-se uma leve distância que se formava do desempenho do Quicksort para os outros dois algoritmos.

Podemos concluir para este caso de inteiros e para as entradas assumidades neste trabalho que o quicksort tenderia a se sair melhor para casos de entrada cada vez melhores. Um gráfico temporal é mostrada abaixo para exemplificar a superioridade do quicksort sobre os outros dois algoritmos nos 3 últimos casos de teste.

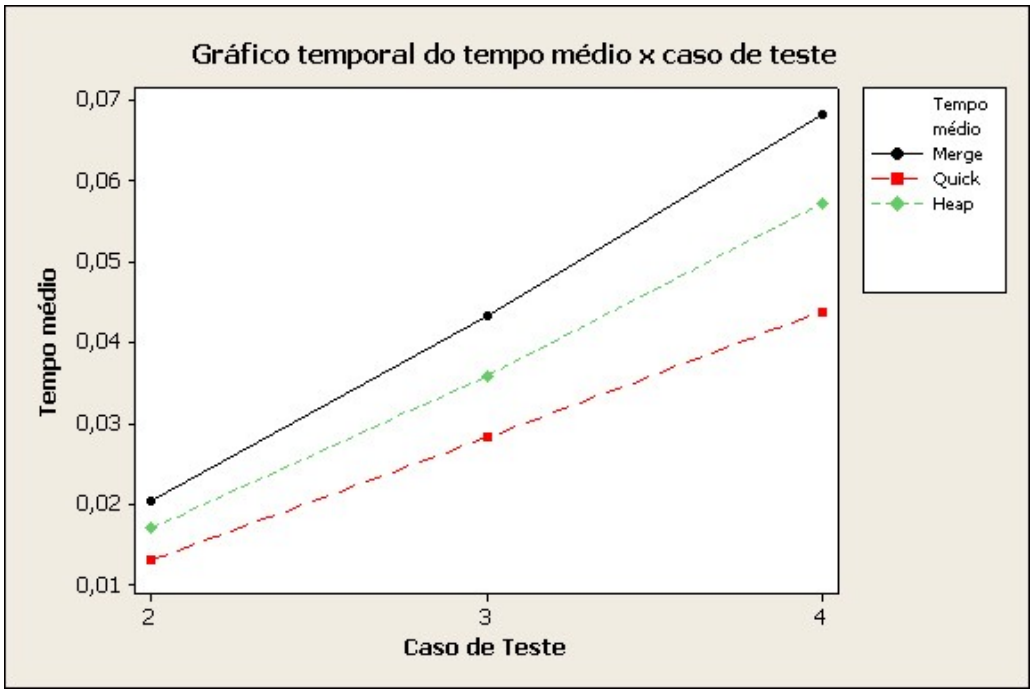


Gráfico 2. Gráfico temporal para cada caso de teste. É percebível o quicksort melhor em vermelho

Caso de teste 2: Binary Int

Esse caso de teste em especial foi o de maior dificuldade pois ele se mostra muito lento para o algoritmo do quicksort. A tabela abaixo demonstra as médias de tempo e seu respectivo desvio padrão:

	Heapsort		Mergesort		Quicksort	
Caso 1 (10 Mil)	0.0016	0.0036	0.0027	0.0044	0.2666	0.0131
Caso 2 (50 Mil)	0.0064	0.0048	0.0166	0.0055	6,67	0,02
Caso 3 (100 Mil)	0.0144	0.0057	0.0354	0.0053	26,67	0,08
Caso 4 (150 Mil)	0.0210	0.0049	0.0546	0.0060	60,15	0,12

Tabela 2. Média e desvio padrão para os casos de teste com binary int.

Um breve histograma para o primeiro caso demonstra a vital diferença do quicksort neste caso de números binários.

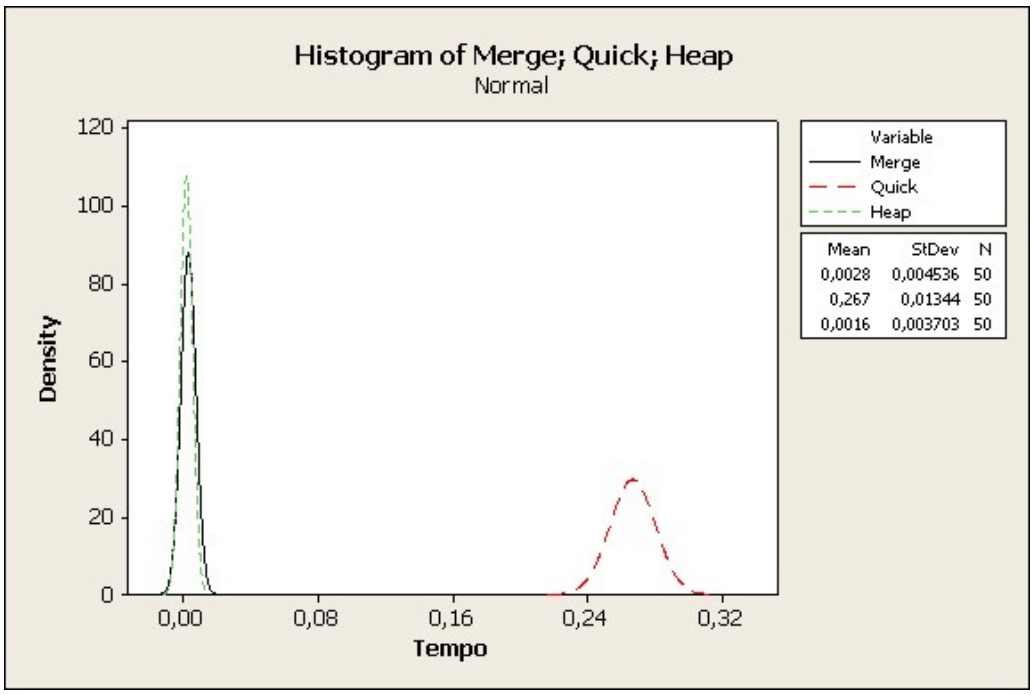


Gráfico 3: Histograma do caso binário para o segundo caso.

O gráfico abaixo mostra o relacionamento do tempo em todos os casos de teste do quicksort para os outros dois algoritmos.

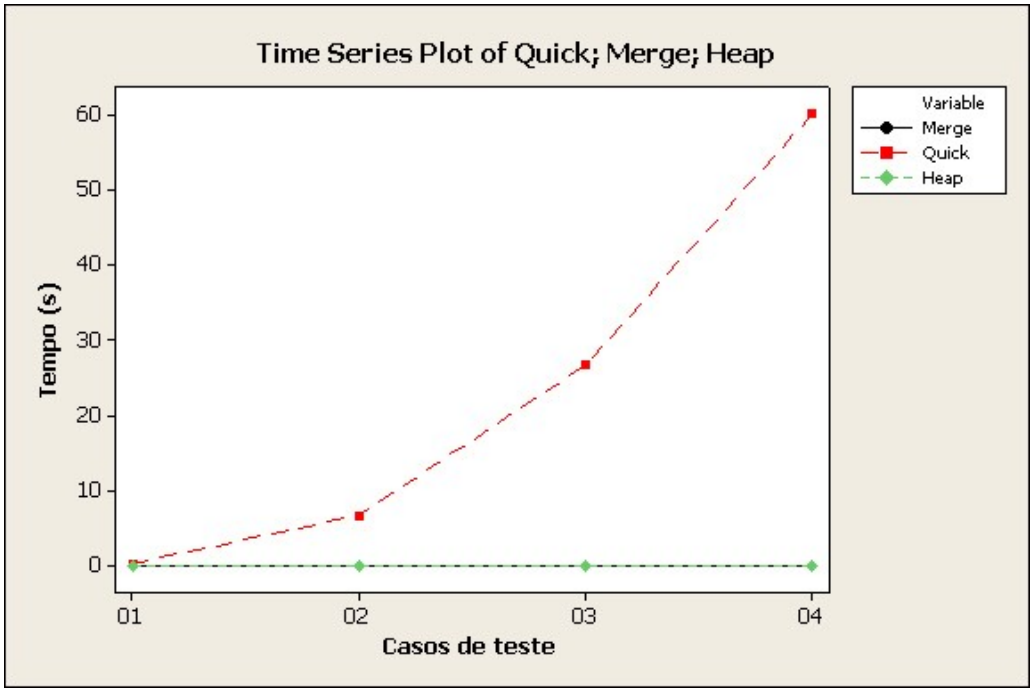


Gráfico 4: Gráfico temporal. Quicksort consome muito mais tempo para ordenar que os outros dois algoritmos.

Isso ocorre devido ao fato de apenas possuir dois elementos, o quicksort acaba caindo no seu pior caso, que por sinal é da ordem de n^2, por isso o tempo tão elevado para ordenação.

Caso de Teste 3: Double Numbers

Os resultados para o números double não foi muito diferente dos que os resultados para short int, mostrando que o caso principal a ser avaliado era Binary Int.

	Heapsort		Mergesort		Quicksort	
Caso 1 (10 Mil)	0.0036	0.0048	0.0037	0.0056	0.0020	0.0040
Caso 2 (50 Mil)	0,16	0.0048	0.0220	0.0056	0.0144	0,01
Caso 3 (100 Mil)	0.0352	0.0049	0.0450	0.0085	0.0272	0,01

Caso 4 (150 Mil)	0.0562	0.0059		0.0683	0.0061	0.0436	0.0082

Tabela 3. Média e desvio padrão para os casos de teste com Double numbers.

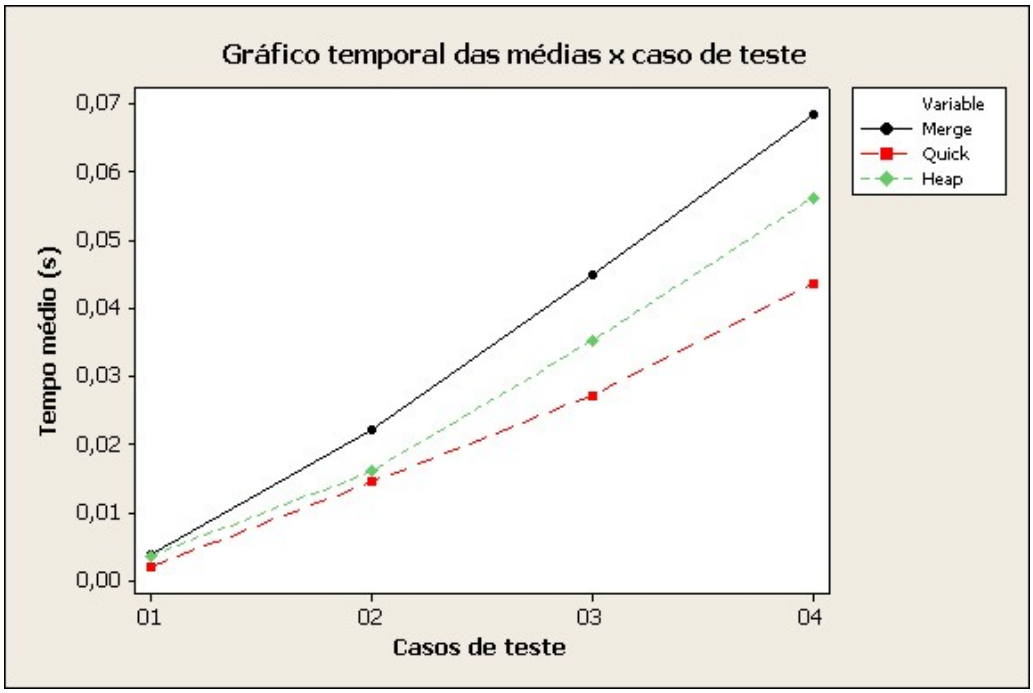


Gráfico 5: Análise do double numbers semelhante ao primeiro caso, o de números inteiros.