Relatório de Análise de Algoritmos

Daniel Marques, Miguel Brito, Jefferson Oliveira, Vinicius Gonzaga June 29, 2017

1 Introdução

1.1 Heap Sort

Analise

O heapsort é um algoritmo de ordenação que se utilizada da estrutura de dados heap. Seu funcionamento consiste em criar um heap de máximo a partir do vetor original, em seguida trocar o primeiro elemento com o último e decrementar o tamanho do heap. Em seguida novamente cria-se um heap de máximo com a nova raiz. Dessa forma segue-se ordenando o vetor, do último para o primeiro elemento.

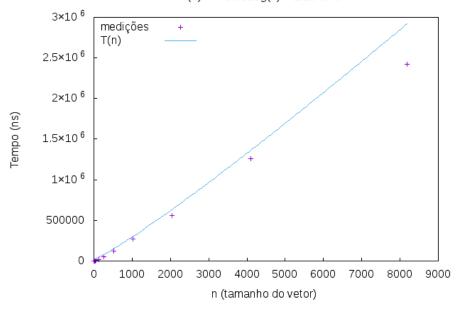
- Tempo no melhor caso: $\theta(nlgn)$
- Tempo no pior caso: $\theta(nlgn)$
- Tempo no caso médio: $\theta(nlgn)$

2 Tempos

2.1 Vetores aleatórios

Figure 1: Heapsort vetor aleatório

Método HeapSort com vetor aleatorio T(n) = n*39.3Log(n) + 2.294e+04



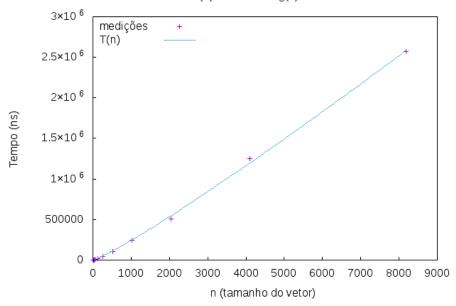
Para $n = 2^{32}$:

 $T(n) = 2^{32} * 39.3 * lg 2^{32} + 22940 = 5.4013509e + 12 ns = 90,02 min.$

2.2 Vetores totalmente crescentes

Figure 2: Heapsort vetor crescente

Método HeapSort com vetor crescente T(n) = n*35.05Log(n) + 0.46



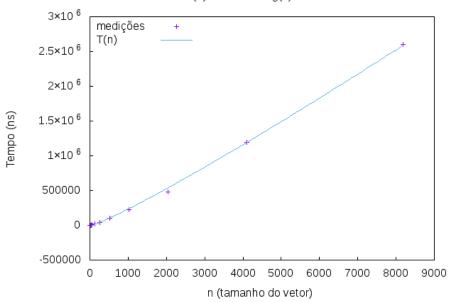
Para $n = 2^{32}$:

 $T(n) = 2^{32} * 35.05 * lg \ 2^{32} + 0.46 = 4.8172353e + 12 \ ns = 80,28 \ min.$

2.3 Vetores totalmente decrescentes

Figure 3: Heapsort vetor decrescente

Método HeapSort com vetor decrescente T(n) = n*35.24Log(n) + .9700



Para $n = 2^{32}$:

 $T(n) = 2^{32} * 35.24 * lg 2^{32} - 9700 = 4.8433487e + 12 ns = 80,72 min.$

2.4 Vetores parcialmente crescentes

Figure 4: Heapsort vetor parcialmente crescente 60 %

Método HeapSort com vetor parcialmente crescente(60) T(n) = n*36.76Log(n) + 63253×10 ⁶ medições T(n) 2.5×10⁶ 2×10 6 Tempo (ns) 1.5×10⁶ 1×10 ⁶ 500000 0 1000 4000 5000 6000 0 2000 3000 7000 8000 9000 n (tamanho do vetor)

Figure 5: Heapsort vetor parcialmente crescente 70 %

Método HeapSort com vetor parcialmente crescente(70)

T(n) = n*37.58Log(n) + 80843×10 ⁶ medições T(n) 2.5×10⁶ 2×10 ⁶ Tempo (ns) 1.5×10⁶ 1×10 ⁶ 500000 0 1000 2000 3000 4000 5000 6000 7000 8000 n (tamanho do vetor)

Figure 6: Heapsort vetor parcialmente crescente 80 %

Método HeapSort com vetor parcialmente crescente(70) T(n) = n*37.58Log(n) + 80843×10 ⁶ medições T(n) 2.5×10⁶ 2×10 ⁶ Tempo (ns) 1.5×10⁶ 1×10 ⁶ 500000 0 0 1000 2000 3000 4000 5000 6000 7000 8000 9000

Figure 7: Heapsort vetor parcialmente crescente 90 %

n (tamanho do vetor)

Método HeapSort com vetor parcialmente crescente(70)

T(n) = n*37.58Log(n) + 80843×10⁶ medições T(n) 2.5×10⁶ 2×10 ⁶ Tempo (ns) 1.5×10⁶ 1×10 ⁶ 500000 0 1000 0 2000 3000 4000 5000 6000 7000 8000 9000 n (tamanho do vetor)

2.5 Vetores parcialmente decrescentes

Figure 8: Heapsort vetor parcialmente decrescente 60 %

Método HeapSort com vetor parcialmente decrescente(60) T(n) = n*36.76Log(n) + 1.553

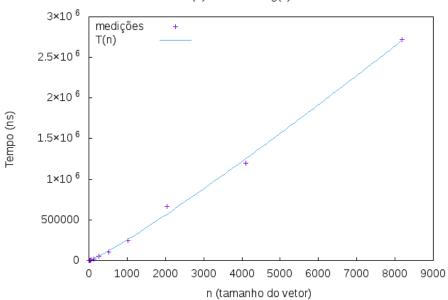


Figure 9: Heapsort vetor parcialmente decrescente 70 %

Método HeapSort com vetor parcialmente decrescente(60) T(n) = n*36.76Log(n) + 1.553

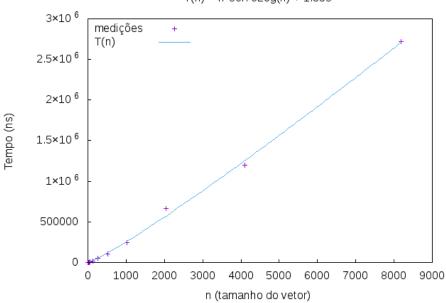


Figure 10: Heapsort vetor parcialmente decrescente 80 %

Método HeapSort com vetor parcialmente decrescente(60) T(n) = n*36.76Log(n) + 1.553

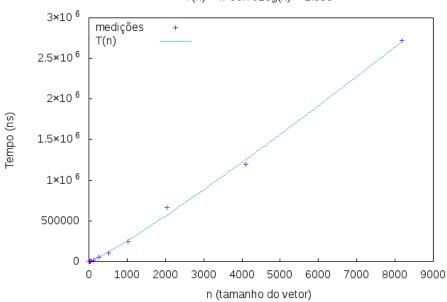


Figure 11: Heapsort vetor parcialmente decrescente 90 %

Método HeapSort com vetor parcialmente decrescente(60) T(n) = n*36.76Log(n) + 1.553

