Algoritmos e Estruturas de Dados I

Professor: André Carlos Ponce de Carvalho

Algoritmos de Ordenação

Análise assintótica, de memória, e estatística de algoritmos de ordenação

(Bubble, Selection, Quick e Heap Sort)

Aluno: Felipe Barbosa de Oliveira

Nº USP: 10276928

INTRODUÇÃO

Neste trabalho, foi analisada a complexidade de quatro algoritmos de ordenação de vetores: o Quick Sort, o Heap Sort, o Bubble Sort e o Selection Sort. Também foi discutido o uso de memória e o funcionamento de cada um deles. Além disso, os algoritmos foram testados usando vetores desordenados de tamanho 10 até 100000, utilizando de um contador para se ter uma noção da sua curva de complexidade.

ANÁLISE ASSINTOTICA

Bubble Sort:

O algoritmo com o contador foi implementado da seguinte maneira, utilizando a mesma lógica de posicionamento do contador dos trabalhos anteriores:

```
void bubble_sort (int vetor[], int n) {
    int k, j, aux;

    for (k = n - 1; k > 0; k--) {
        for (j = 0; j < k; j++) {

            if (vetor[j] > vetor[j + 1]) {
                aux = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = aux;

                contagemBubble++;

            }
        }
    }
}
```

Vemos que o algoritmo inteiro se resume a um procedimento dentro de duas estruturas de repetição, sendo que cada uma repete um número linear a n de vezes, sendo n o tamanho do vetor. Portanto, podemos facilmente ver que o algoritmo é sempre $O(n^2)$.

Selection Sort:

O algoritmo foi implementado do seguinte modo:

```
void selection_sort(int num[], int tam) {
    int i, j, Min, aux;
    for (i = 0; i < (tam-1); i++){
        Min = i;

        for (j = (i+1); j < tam; j++) {
            if(num[j] < num[Min]){
                Min = j;

                contagemSelection++;
            }
        }

        if (num[i] != num[Min]) {
            aux = num[i];
            num[i] = num[Min];
            num[Min] = aux;
        }
    }
}
```

Vemos uma estrutura muito parecida com o Bubble Sort, com uma estrutura de repetição dentro de outra, com ambas lineares a n. Porém desta vez, possui uma condição apenas dentro da estrutura de repetição de fora, que repete n vezes. Portanto, podemos logicamente afirmar que o algoritmo é sempre $O(n^2 + n)$, que pode ser simplificado a $O(n^2)$.

Quick Sort:

O algoritmo com o contador se encontra ao lado. Por se tratar de um algoritmo recursivo, utilizaremos o método de árvore de recorrência para resolver os casos. Temos que a função f da equação de recorrência será n, pois se trata basicamente de um while principal dependente de n com dois whiles dentro dele, porém sendo muito

```
void quick_sort(int vetor[], int inicio, int fim){

    int pivo, aux, i, j, meio;

    i = inicio;
    j = fim;

    meio = (int) ((i + j) / 2);
    pivo = vetor[meio];

    do{
        while (vetor[i] < pivo){
            i = i + 1;
            contagemQuick++;
        }

        while (vetor[j] > pivo){
            j = j - 1;
            contagemQuick++;
        }

        if(i <= j){
            aux = vetor[i];
            vetor[i] = vetor[j];
            vetor[j] = aux;
            i = i + 1;
            j = j - 1;
        }

    }while(j > i);

    if(inicio < j) quick_sort(vetor, inicio, j);
    if(i < fim) quick_sort(vetor, i, fim);
}
```

instáveis dependendo do vetor, e de poucas repetições. O pior caso se dá quando o pivô é o menor ou o maior número do vetor, criando duas separações muito desbalanceadas. Deste modo, a recursividade chamará um vetor de tamanho $n - 1$. Então temos:

$$T(n) = T(n-1) + n$$

Que resolvendo pelo método de árvore (recorrência já resolvida no trabalho anterior), resultamos em $T(n) = O(n^2)$.

Já no melhor caso, temos que o pivô separa o vetor em dois vetores de tamanhos iguais (ou diferentes apenas de 1 caso n seja ímpar). Assim, temos:

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

Que, também resolvendo pelo método de árvore (recorrência também resolvida no trabalho anterior), resultamos em $T(n) = O(n * logn)$.

Heap Sort:

Foi implementado da seguinte maneira:

```
void heapify(int arr[], int n, int i){
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    if (l < n && arr[l] > arr[largest]){
        largest = l;
        contagemHeap++;
    }

    if (r < n && arr[r] > arr[largest]){
        largest = r;
        contagemHeap++;
    }

    if (largest != i){
        Swap(&(arr[i]), &(arr[largest]));
        contagemHeap++;
        heapify(arr, n, largest);
    }
}
void heap_sort(int arr[], int n){

    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i=n-1; i>=0; i--){
        Swap(&(arr[0]), &(arr[i]));
        heapify(arr, i, 0);
    }
}
```

RESULTADOS E DISCUSSÕES

Bubble Sort:

Os valores encontrados nos testes foram:

Tamanho	1	2	3	4	5	6	7	8	9	10
10	27	19	26	20	21	19	26	17	10	28
100	2671	2514	2399	2062	2622	2225	2347	2258	2506	2581
1000	239516	257220	251770	242152	249760	249506	249182	234767	246963	247685
10000	25108513	24896633	25124229	24640478	25153070	25074665	24804792	24655251	24982351	24870787
100000	2496398352	2501911150	2511206208	2497445118	2499770788	2503395668	2506745764	2511188300	2494269154	2499629624

E com isso, foram feitas as seguintes estatísticas:

Tamanho	Media	Mediana	Maior	Menor
10	21,3	20,5	28	10
100	2418,5	2452,5	2671	2062
1000	246852,1	248433,5	257220	234767
10000	24931076,9	24939492	25153070	24640478
100000	2502196013	2500840969	2511206208	2494269154

Vemos facilmente que o número de contagens cresce de maneira exponencial, o que está de acordo com o fato do algoritmo ser $O(n^2)$. Vemos também que se trata de um algoritmo não muito estável, com grandes variações em menores tamanhos de vetor.

Selection Sort:

Tamanho	1	2	3	4	5	6	7	8	9	10
10	13	10	10	10	11	10	10	8	10	9
100	305	310	313	316	307	294	265	318	357	318
1000	5086	5237	5054	5334	5241	5081	5412	5259	5368	5549
10000	74983	75583	75988	74825	74588	76197	76165	76011	75356	75289
100000	942338	944796	946533	945623	947828	948841	939251	942505	953939	952069

Tamanho	Media	Mediana	Maior	Menor
10	10,1	10	13	8
100	310,3	311,5	357	265
1000	5262,1	5250	5549	5054
10000	75498,5	75469,5	76197	74588
100000	946372,3	946078	953939	939251

Novamente, podemos ver o tempo cresce de maneira exponencial, porém com valores muito menores que do Bubble Sort. O algoritmo é bem mais estável e com crescimento exponencial bem pequeno.

Quick Sort:

Tamanho	1	2	3	4	5	6	7	8	9	10
10	18	13	17	12	20	15	20	17	19	28
100	443	392	392	448	473	438	531	551	365	360
1000	7358	7922	7260	7936	7496	9095	7181	7607	7144	7290
10000	99208	126384	101045	108179	93924	96347	93921	94490	111037	105676
100000	1249030	1252077	1282391	1305382	1248382	1311485	1391969	1277037	1218587	1365486

Tamanho	Media	Mediana	Maior	Menor
10	17,9	17,5	28	12
100	439,3	440,5	551	360
1000	7628,9	7427	9095	7144
10000	103021,1	100126,5	126384	93921
100000	1290182,6	1279714	1391969	1218587

No caso do Quick Sort, vemos que ele cresce de maneira ainda mais suave que o Selection Sort, o que faz sentido contado o fato de ser $O(n * logn)$. A variância de resultados é relativamente grande, principalmente para valores menores, e se mostra um algoritmo muito mais rápido e eficiente que os dois anteriores (contando que o número de contadores neste é o dobro que nos anteriores).

Heap Sort:

Tamanho	1	2	3	4	5	6	7	8	9	10
10	34	41	47	44	28	32	36	28	38	40
100	1116	1137	1177	1140	1146	1137	1176	1123	1178	1119
1000	19758	19799	19836	19680	19767	19732	19704	19804	19833	19625
10000	280517	280578	280588	280457	280626	280453	280553	280454	280816	280966
100000	3638122	3637644	3638367	3637275	3637526	3638971	3638640	3638074	3638414	3638656

Tamanho	Media	Mediana	Maior	Menor
10	36,8	37	47	28
100	1144,9	1138,5	1178	1116
1000	19753,8	19762,5	19836	19625
10000	280600,8	280565,5	280966	280453
100000	3638168,9	3638244,5	3638971	3637275

O algoritmo mostrou-se menos eficiente que o Quick Sort (mesmo considerando o fato de ter um contador a mais). O crescimento é maior que o do Quick Sort, porém mais suave que os dois primeiros (também sendo coerentes com o fato de ser da mesma complexidade do Quick Sort).

MEMÓRIA

Podemos facilmente inferir que tanto o Bubble Sort quanto o Selection Sort são os menores consumidores de memória dentre os quatro algoritmos. Isso se dá pelo fato de serem os únicos iterativos, não criando novas variáveis a cada repetição. Já o Quick Sort e o Heap Sort, por serem recursivos, irão alocar mais memória pois irão criar novas variáveis a cada recorrência. Portanto a complexidade espacial do Bubble Sort e do Selection Sort teóricamente é sempre a mesma, não dependendo do tamanho do vetor. Já nos outros dois, a complexidade dependerá de quantas recorrências ocorrerem, ou seja, depende diretamente do tamanho do vetor e de seu grau de organização.

INVERSÃO DOS ALGORITMOS

Os algoritmos na sua forma inversa (ordenando de maneira decrescente) foram implementados no código junto com este relatório.

CONCLUSÕES

Com isso, concluímos que tanto o Bubble como o Selection Sort são os dois algoritmos mais lentos em quase todos os casos (principalmente o Bubble Sort), porém, consomem consideravelmente menos memória, podendo então ser viáveis em situações onde o tempo é menos importante que o espaço. Já o Quick e o Heap Sort se tratam do

contrário. Por serem recursivos, alocam muito mais memória que os outros dois, porém, mostraram-se muito mais rápidos e de complexidade menor. O algoritmo mais rápido dentre estes, foi o Quick Sort.