UNIVERSIDADE FEDERAL DE OURO PRETO

BCC 202 – Estrutura de Dados 1

Trabalho Pratico 3.

Professora: Andrea

Aluno: André Ribeiro de Brito

Matricula: 11.2.4985

INTRODUÇÃO

O Heap Sort é um tipo de algoritmo de ordenação, onde seu funcionamento r através de uma estrutura de dados, onde ordenamos os elementos a medida que os insere na estrutura, assim no final das inserções os elementos podem ser removidos da raiz do algoritmo, na ordem desejada. O Heap pode ser representável através de uma arvore ou como vetor.

A solução do problema a ser tratado é uma criação do algoritmo de ordenação heap sort onde foi trabalhado com vetores de 100, 500, 1000, 5000, e 10000, onde foi comparado números de troca, copia e o tempo de execução desses vetores com inserções aleatório, ordenado, inversamente e quase ordenado.

Quick sort é um tipo de algoritmo de ordenação, onde tem como principio escolher um valor médio do vetor, para que todos os valores maiores do que ele passa a frente e todos os menores para trás, ou seja, dividir e conquistar, primeiramente ele divide uma grande lista em dois pequenos sub-listas: os elementos baixos e os elementos elevados, então recursivamente classificar as sublistas.

A solução do problema a ser tratado é uma criação do algoritmo de ordenação quick sort utilizando o método de inserção trabalhando com vetores de 100, 500, 1000, 5000, e 10000, onde foi comparado números de troca, copia e o tempo de execução desses vetores com inserções aleatório, ordenado, inversamente e quase ordenado.

Implementação

Estrutura de dados utilizas no .h onde foi colocado todas as funções que o código ira conter.

```
typedef long Chave;

typedef struct {

    Chave chave;

}Registro;

void Heap_Refaz(Registro* V, int , int );

void Heap_Constroi(Registro* V, int n);

void Heap_Aleatorio(Registro* V, int n);
```

```
void Heap_Ordenado(Registro* V, int n);

void Heap_Inversamente(Registro* V, int n);

void Heap_QuaseOrdenado(Registro* V, int n);



void QuickSortAleatorio(Registro* V, int n);

void QuickSortOrdenado(Registro* V, int n);

void QuickSortInversamente(Registro* V, int n);

void QuickSortQuaseOrdenado(Registro* V, int n);

void QuickSort_ordena(Registro* V, int esq, int dir);

void QuickSort_particao(Registro* V, int esq, int dir, int *i, int *j);

void Insercao (Registro* V, int n);

void imprimir(Registro* V, int n);
```

Estruturas de dados do .c onde contem as 8 função dos valores a serem gerados e outras função de que contem no método de ordenação.

```
#include <stdio.h>

#include <stdlib.h>

#include "ordenacao.h"

#include <time.h>

//#define TAM 100;

void Heap_Refaz(Registro* V, int esq, int dir ){

    int i = esq;

    int j= (i * 2) + 1;

    Registro aux;

    aux = V[i];

    while (j <= dir) {
```

```
        if ((j < dir) && (V[j].chave < V[j+1].chave))
            j++; // j recebe o outro filho de i
        if(aux.chave >= V[j].chave)
            break; // heap foi refeito corretamente
         V[i] = V[j];
         i = j;
         j = i*2 + 1; // j = primeiro filho de i
     }
     V[i] = aux;
 }
void Heap_Constroi(Registro* V, int n) {
    int esq;
    esq = (n/2) - 1;
    while (esq >= 0) {
        Heap_Refaz(V,esq,n - 1);
        esq--;
}
}
void Heap_Aleatorio(Registro* V, int n){
    Registro ax;
   clock_t ini1, time; // tempo de inicio
   ini1=clock();
   //printf("HeaP aleatorio\n");
    Heap_Constroi(V, n);
```

```
    while (n > 1) {

        ax = V[n-1];

        V[n-1]= V[0];

        V[0] = ax;

        n--;

        Heap_Refaz(V, 0, n-1); // refaz o heap

        printf("\n");

        imprimir(V, n);

}

    printf("duracao do metodo aleatorio e: %f milisegundo",((double)( time - ini1 )
/ ((double)CLOCKS_PER_SEC )));;



    }

void Heap_Ordenado(Registro* V, int n) {

    Registro aux6;

    clock_t ini2, time2; // tempo de inicio

    ini2=clock();

    Heap_Constroi(V, n);

    while (n > 1) {

        aux6 = V[n-1];

        V[n-1] = V[0];

        V[0] = aux6;

        n--;

        Heap_Refaz(V, 0, n-1);// refaz o heap

        imprimir(V, n);
```

```
 }

     printf("duracao do metodo ordenado e: %f",((double)( time2 - ini2 ) /
((double)CLOCKS_PER_SEC )));

}

void Heap_Inversamente(Registro* V, int n){

    Registro aux1;

    clock_t ini3, time3; // tempo de inicio

    ini3=clock();

    Heap_Constroi(V, n);

    while (n < 1) {

        aux1 = V[0];

        V[0] = V[n-1];

        V[n-1] = aux1;

        n++;

        Heap_Refaz(V, 0, n-1); // refaz o heap

    imprimir(V, n);

 }

    printf("duracao do metodo inversamente e: %f",((double)( time3 - ini3 ) /
((double)CLOCKS_PER_SEC )));

}

void Heap_QuaseOrdenado(Registro* V , int n){

    Registro aux;

    clock_t ini4, time4; // tempo de inicio

    ini4=clock();

    Heap_Constroi(V, n);
```

```
    while (n > 1) {

        aux = V[n-1];

        V[n-1] = V[0];

        V[0] = aux;

        n--;

        Heap_Refaz(V, 0, n-1); // refaz o heap

        imprimir(V, n);

}

    printf("duracao do metodo quase ordenado e: %f",((double)( time4 - ini4 ) /
((double)CLOCKS_PER_SEC )));

   // printf("\n\n");

}

void QuickSortAleatorio(Registro *V, int n) {

   clock_t ini5, time5; // tempo de inicio

   ini5=clock();

   QuickSort_ordena(V, 0, n-1);

   imprimir( V, n);

   printf("duracao do metodo quase ordenado e: %f",((double)( time5 - ini5 ) /
((double)CLOCKS_PER_SEC )));

}

void QuickSortOrdenado(Registro *V, int n){

    clock_t ini6, time6; // tempo de inicio

    ini6=clock();

    QuickSort_ordena(V, 0, n-1);

    imprimir(V,  n);

    printf("duracao do metodo quase ordenado e: %f",((double)( time6 - ini6 ) /
```

```
((double)CLOCKS_PER_SEC )));

}

void QuickSortInversamente(Registro *V, int n){

    clock_t ini7, time7; // tempo de inicio

    ini7=clock();

    QuickSort_ordena(V, n-1, 0);

    imprimir(V, n);

    printf("duracao do metodo quase ordenado e: %f",((double)( time7 - ini7 ) /
((double)CLOCKS_PER_SEC )));

}

void QuickSortQuaseOrdenado(Registro *V, int n){

    clock_t ini8, time8; // tempo de inicio

    ini8=clock();

    QuickSort_ordena(V, 0, n-1);

    imprimir( V, n);

    printf("duracao do metodo quase ordenado e: %f",((double)( time8 - ini8 ) /
((double)CLOCKS_PER_SEC )));

}

void QuickSort_ordena(Registro* V, int esq, int dir) {

  int i, j, n;

  if(dir - esq < 10)

     Insercao (V, n);

    QuickSort_particao(V, esq, dir, &i, &j);

      if (esq < j)

          QuickSort_ordena(V, esq, j);
```

```
      if (i < dir)
          QuickSort_ordena(V, i, dir);
    }
void QuickSort_particao(Registro *v, int esq, int dir, int *i, int *j) {
    Registro pivo;
    Registro aux;
     *i = esq; *j = dir;
    pivo = v[(*i + *j)/2]; /* obtem o pivo x */
    do {
    while (pivo.chave > v[*i].chave) (*i)++;
    while (pivo.chave < v[*j].chave) (*j)--;
    if (*i <= *j) {
     aux = v[*i];
     v[*i] = v[*j];
     v[*j] = aux;
     (*i)++; (*j)--;
     }
     }
    while (*i <= *j);
}
void Insercao (Registro* V, int n) {
    int i,j;
    for (i = 0; i <50; i++) {
        V[n] = V[i];
```

```
        j = i + 1;
        while (V[n].chave > V[j].chave) {
            V[j - 1] = V[j];
            j++;
            }
        V[j - 1] = V[n];
        }
}
void imprimir(Registro *V, int n){
    int i;
    for (i = 0; i < n; i++){
        printf("%2d ",V[i].chave);
    }
    printf("\n");
 }
```

E por fim o programa principal que contem os método de gera o vetor aleatório, gerado pela função rand() e outros métodos que coloca o vetor ordenado, inversamente ordenado e o quase ordenado que foi utilizado uma variável auxiliar que nela contem uma parte do vetor aleatório mais uma outra parte do vetor ordenado para esta gerando o vetor aleatório.

```
int main(int argc, char** argv) {
    Registro V[100];
    Registro aux[100];
    Registro aux1[100];
    Registro aux3[100];
    int i;
```

```
  srand(time(NULL));
  for(i=0; i<TAM;i++){
      V[i].chave=rand()%100;
}
  for(i=0;i<TAM;i++){
      aux[i].chave = i;
      aux3[i].chave = (V[i].chave + aux[i].chave)/2;
  }
  for(i=TAM; i > 0;i--){
      aux1[i].chave=i;
  }
  printf("Aleatorio\n");
  Heap_Aleatorio(V ,10);
  printf("\n");
  printf("Ordenado\n");
  Heap_Ordenado(aux ,10);
  printf("\n");
  printf("Inversamente\n");
  Heap_Inversamente(aux1 ,10);
  printf("\n");
  printf("Quase Ordenado\n");
  Heap_QuaseOrdenado(aux3 ,10);
  printf("\n");
```

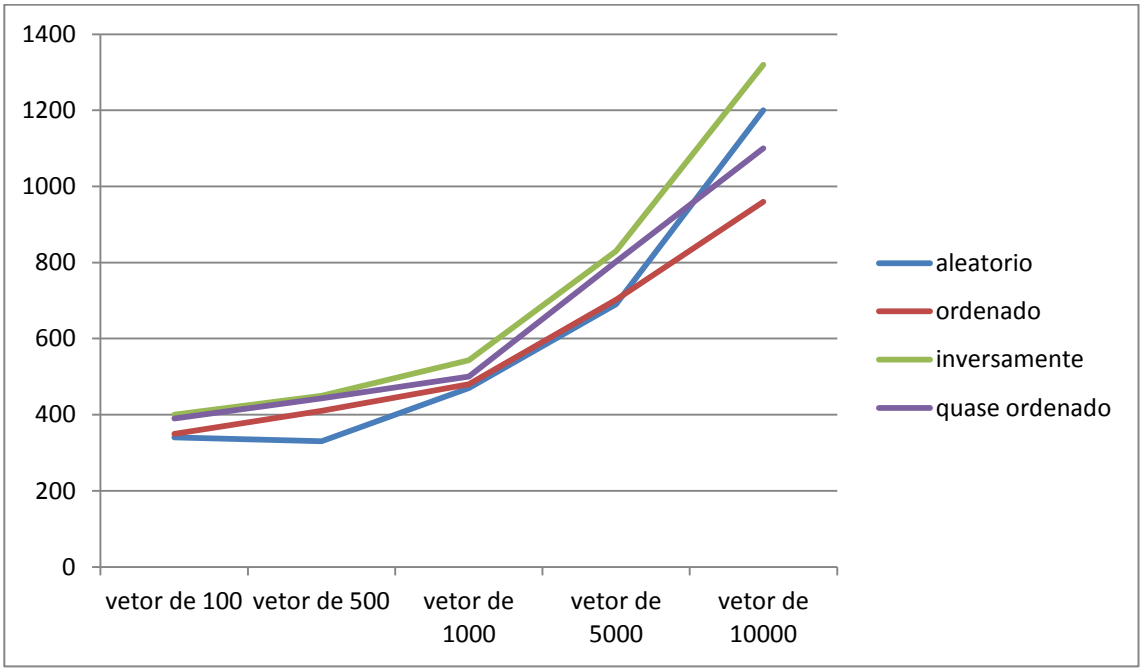
```
printf("Aleatorio do quicksort\n");
QuickSortAleatorio(V ,100);
  imprimir(V , 100);
printf("\n");
printf("Quase Ordenado do quicksort\n");
QuickSortOrdenado(aux ,100);
imprimir(V , 100);
printf("\n");
printf("Inversamente Ordenado do quicksort\n");
QuickSortInversamente(aux1 ,100);
 imprimir(V , 100);
printf("\n");
printf("Quase Ordenado do quicksort\n");
QuickSortQuaseOrdenado(aux3 ,100);
imprimir(V , 100);
```

Complexidade

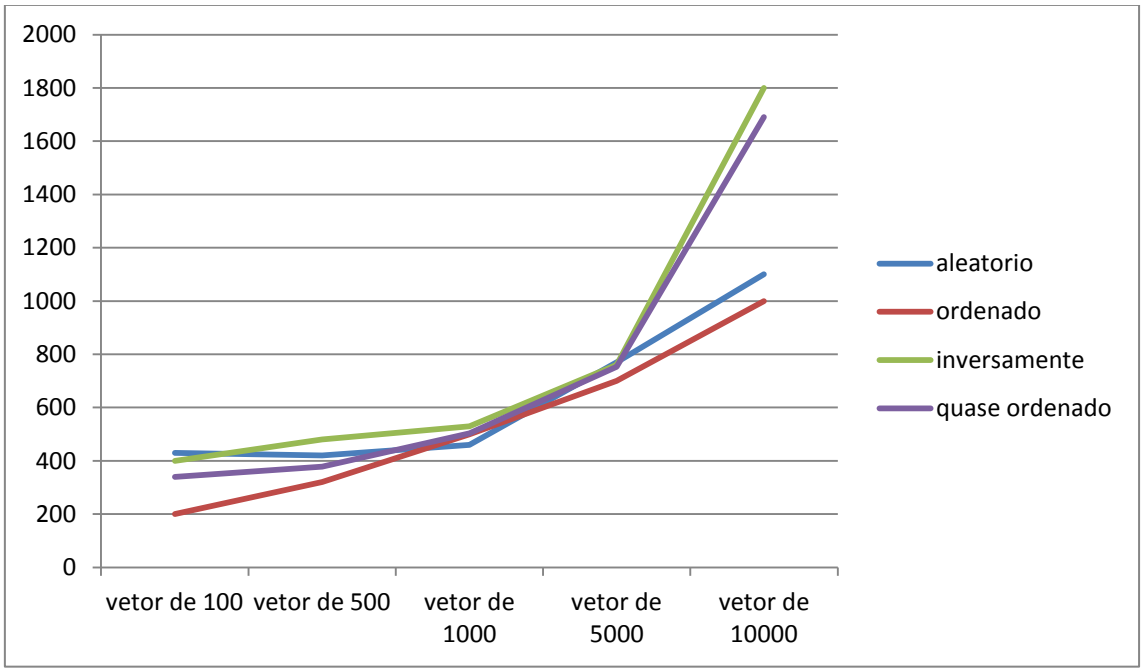
Sua complexidade do heap sort foi de O(n log n) para todos os casos.
Já o algoritmo quick sort o pior caso é $O(n^2)$, melhor caso O(n log n) e no caso médio também O(n log n).

Teste Realizados

A tabela a seguir mostra o desempenho de tempo do algoritmo heap sort em milissegundos.



ja o gráfico da algoritmo quick sort em relação tamanho e tempo por milissegundo ficou:



Conclusão

Foram realizados vários teste mais sempre quando repetia dava um intervalo de tempo muito alto sendo com o mesmo vetor e o tipo.

Achei esse trabalho interessante apesar da dificuldade que tive na parte do quick sort com inserção e colocar a função tempo para calcular.

Foi pesquisado para fazer o trabalho materiais do prof. Túlio Toffolo e busca em site de enciclopédia .