# **UNIVERSIDADE FEDERAL DA PARAÍBA**

# **CENTRO DE INFORMÁTICA**

Disciplina: Análise e Projeto de Algoritmos

Entrega da atividade: 19/05/2020

Professor: Gilberto

Disciplina: Estrutura de Dados – Mestrado 2020.1

Aluno: Anderson Ernani de Oliveira

**Ordenação por Comparação 3**

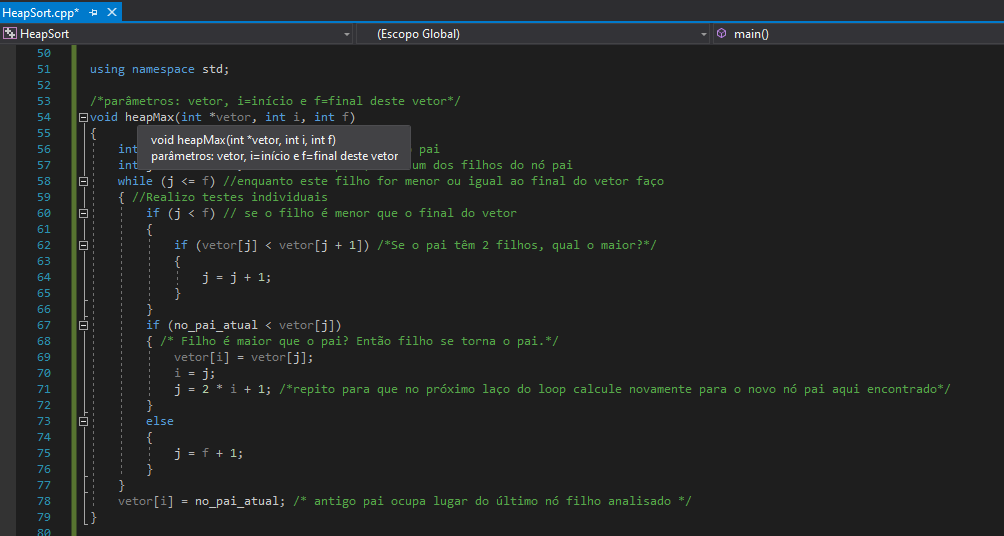
Implemente as seguintes estruturas e algoritmos:

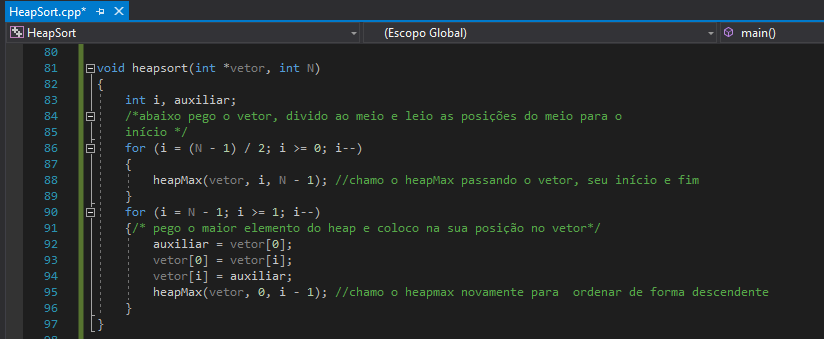
* Heap máximo
* HeapSort

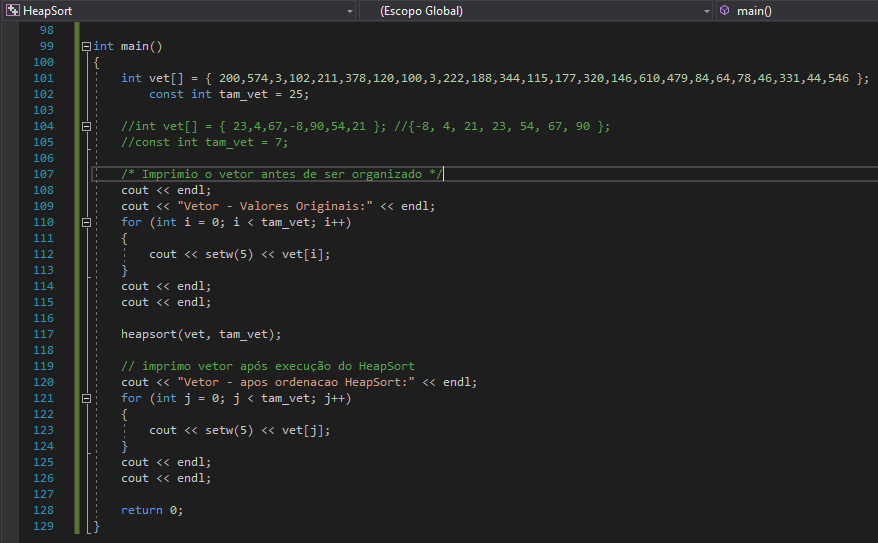
Abaixo código fonte em c++ e print de tela do programa após execução.

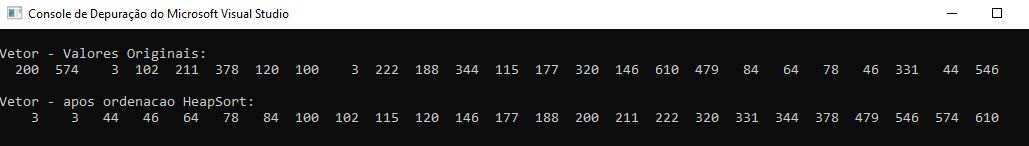
Vetor que representa a árvore binária antes da ordenação

{200, 574, 3, 102, 211, 378, 120, 100, 3, 222, 188, 344, 115, 177, 320, 146, 610, 479, 84, 64, 78, 46, 331, 44, 546}.









/\*16-05-2020 Anderson Ernani de Oliveira

Mestrado UFPB - Estrutura de Dados - Heap Sort - prof. Gilberto

\*/

/\*

Um Heap A é um objeto com dois

atributos:

– comprimento[A] : que é o número possível de elementos no arranjo;

– tamanho\_heap[A] : o número de elementos no heap.

onde,

tamanho\_heap[A] ≤ comprimento[A]

A raiz de um heap binário é A[1], o primeiro elemento.

• Dado um elemento i do heap temos:

– PAI(i) = i/2;

– ESQUERDO(i) = 2i;

– DIREITO(i) = 2i + 1;

Heap Máximo e Mínimo

Em um Heap Máximo, todo nó i diferente da raiz tem:

A[PAI(i)] ≥ A[i] Ou seja, o valor de um nó i é no máximo o valor de seu pai, logo, o maior elemento

de um heap máximo está na raiz. O heap mínimo é organizado de forma oposta:

A[PAI(i)] ≤ A[i].

Construção de um Heap

• Os elementos do subarranjo A[(└ n/2 ┘+1) .. n )] , sendo n = tamanho\_heap[A], são todos folhas da árvore e então cada um

deles é um heap máximo de 1 elemento.

• Para construir um heap máximo a partir de um arranjo A, o procedimento BUILDMAX-HEAP percorre os nós restantes da

árvore e executa MAX-HEAPIFY sobre cada um de baixo para cima.

Algoritmo HeapSort

• O algoritmo HeapSort começa usando BUILDMAX-HEAP para construir um heap no arranjo

de entrada A[1 .. n].

• Tendo em vista que o elemento máximo do arranjo está na raiz A[i], ele pode ser colocado

em sua posição final correta, trocando-se esse elemento por A[n].

• Se agora diminuirmos o valor de tamanho\_heap[A] podemos transformar o arranjo A novamente em um heap máximo

chamado MAX\_HEAPIFY(A, 1) .

• O algoritmo HeapSort repete este processo até que o arranjo tenha apenas um elemento.

Vídeo pesquisado https://www.youtube.com/watch?v=JJRrzo36Vlg

https://www.youtube.com/watch?v=zSYOMJ1E52A

\*/

#include <iostream>

#include <iomanip>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <chrono>

using namespace std;

/\*parâmetros: vetor, i=início e f=final deste vetor\*/

void heapMax(int \*vetor, int i, int f)

{

int no\_pai\_atual = vetor[i]; //guardo o nó pai

int j = i \* 2 + 1; //calculo a posição de um dos filhos do nó pai

while (j <= f) //enquanto este filho for menor ou igual ao final do vetor faço

{ //Realizo testes individuais

if (j < f) // se o filho é menor que o final do vetor

{

if (vetor[j] < vetor[j + 1]) /\*Se o pai têm 2 filhos, qual o maior?\*/

{

j = j + 1;

}

}

if (no\_pai\_atual < vetor[j])

{ /\* Filho é maior que o pai? Então filho se torna o pai.\*/

vetor[i] = vetor[j];

i = j;

j = 2 \* i + 1; /\*repito para que no próximo laço do loop calcule novamente para o novo nó pai aqui encontrado\*/

}

else

{

j = f + 1;

}

}

vetor[i] = no\_pai\_atual; /\* antigo pai ocupa lugar do último nó filho analisado \*/

}

void heapsort(int \*vetor, int N)

{

int i, auxiliar;

/\*abaixo pego o vetor, divido ao meio e leio as posições do meio para o

início \*/

for (i = (N - 1) / 2; i >= 0; i--)

{

heapMax(vetor, i, N - 1); //chamo o heapMax passando o vetor, seu início e fim

}

for (i = N - 1; i >= 1; i--)

{/\* pego o maior elemento do heap e coloco na sua posição no vetor\*/

auxiliar = vetor[0];

vetor[0] = vetor[i];

vetor[i] = auxiliar;

heapMax(vetor, 0, i - 1); //chamo o heapmax novamente para ordenar de forma descendente

}

}

int main()

{

int vet[] = { 200,574,3,102,211,378,120,100,3,222,188,344,115,177,320,146,610,479,84,64,78,46,331,44,546 };

const int tam\_vet = 25;

//int vet[] = { 23,4,67,-8,90,54,21 }; //{-8, 4, 21, 23, 54, 67, 90 };

//const int tam\_vet = 7;

/\* Imprimio o vetor antes de ser organizado \*/

cout << endl;

cout << "Vetor - Valores Originais:" << endl;

for (int i = 0; i < tam\_vet; i++)

{

cout << setw(5) << vet[i];

}

cout << endl;

cout << endl;

heapsort(vet, tam\_vet);

// imprimo vetor após execução do HeapSort

cout << "Vetor - apos ordenacao HeapSort:" << endl;

for (int j = 0; j < tam\_vet; j++)

{

cout << setw(5) << vet[j];

}

cout << endl;

cout << endl;

return 0;

}