INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA PIAUÍ

MINISTÉRIO DA EDUCAÇÃO
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CURSO : Analise e desenvolvimento de Sistemas
DISCIPLINA : Estrutura de Dados
PROFESSORA: ELANNE

Nome:_____No.:_____

Em __.12.2023

Prova no 2

1. Sobre as afirmações, associe: (1.0 pt)

(1) Ordena da esquerda para a direita, por ordem crescente ou decrescente. À medida que o vetor vai sendo percorrido ele deixa seus elementos à esquerda ordenados.
(2) Uma das sub-tabelas contém os elementos menores que o pivô enquanto a outra contém os maiores. O pivô é colocado entre ambas, ficando na posição correta.
(3) Compara o elemento com o meio, se o elemento for maior que o meio, vá para a segunda parte do vetor. Se o elemento for menor que o meio, vá para a primeira parte do vetor.
(4) Percorre toda a estrutura comparando o elemento procurado com o elemento da estrutura. Se encontrar, encerra o algoritmo, senão passa para o próximo elemento da estrutura. Se a estrutura tiver n elementos e o elemento procurado não existir na estrutura serão realizadas n comparações (pior caso).

a.( ) 1-pesquisa sequencial 2-quicksort 3-pesquisa binária 4-ordenação por inserção
b.(x) 1-ordenação por inserção 2-quicksort 3-pesquisa binária 4-pesquisa sequencial
c.( ) 1-ordenação por inserção 2-bubblesort 3-pesquisa binária 4-pesquisa sequencial
d.( ) nda

2. Sobre o método abaixo, assinale a alternativa correta.: (1.0 pt)

```
void x(int *lista, int l, int r)
{
   int i, j, p, m;
   i = l;
   j = r;
   m = (l + r) / 2;
   p = lista[m];
   while (i <= j)
   {
      while (lista[i] < p && i < r)
      {
         i++;
      }
      while (lista[j] > p && j > l)
      {
         j--;
      }
      if (i <= j)
      {
```

```
            int aux = lista[i];
            lista[i] = lista[j];
            lista[j] = aux;
            i++;
            j--;
        }
    }
     if (j > l)
    {
        x(lista, l, j);
    }
    if (i < r)
    {
        x(lista, i,r);
    }  }
```

a. ( ) Quicksort, o algoritmo possui, no melhor caso, complexidade $O(n^2)$.
b. ( ) Pesquisa Binária, o algoritmo possui, no melhor caso, complexidade $O(\lg n)$.
c. ( ) Pesquisa Binária, o algoritmo possui, no melhor caso, complexidade $O(n^2)$.
d. (x) Quicksort, a complexidade do algoritmo pode variar dependendo do caso. Em média, o Quicksort tem uma complexidade de tempo $O(n \log n)$, que é considerado o melhor caso e ocorre quando o pivô divide o array em duas partes quase iguais. Nesse cenário, cada chamada recursiva processa um array com metade do tamanho, resultando em um tempo de execução eficiente. No pior caso a complexidade de tempo é $O(n^2)$.

3. Verifique o código abaixo e MARQUE A ALTERNATIVA VERDADEIRA: (1.0 pt)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
class No{
        public:
                int mat;
                char nome[23];
                No *prox;
                No *ant;
                No(int m,char n[23]){
                        mat=m;
                        strcpy(nome,n);
                        prox=NULL;
                        ant=NULL;
                }
};
class Lista{
        public:
                No *inicio;
                No *fim;
```

```
Lista(){
        inicio = NULL;
        fim = NULL;
        }
    void x(int m, char n[23]){
                No *novo= new No(m,n);
                if (fim==NULL){
                        inicio = novo;
                        fim = novo;
                }
                else{
                        fim->prox=novo;
                        novo->ant=fim;
                        fim=novo;
                }
        }

        void y(int m, char n[23]){
                No *novo= new No(m,n);
                if (fim==NULL){
                        inicio = novo;
                        fim = novo;
                }
                else{
                        novo->prox=inicio;
                        inicio->ant=novo;
                        inicio=novo;
                }
        }
}
```

a.( ) Trata-se de uma lista simplismente encadeada e o metodo y inclui no inicio da lista
b.( ) Trata-se de uma lista simplismente encadeada e o metodo x inclui no inicio da lista
c.(x) Trata-se de uma lista duplamente encadeada e o metodo y inclui no inicio da lista
d.( ) Trata-se de uma lista duplamente encadeada e o metodo x inclui no inicio da lista

4. Sobre uma pesquisa linear é correto afirmar: (1.0 pt)

A.(v) O melhor caso se dá quando o elemento procurado está na primeira posição da estrutura (p é igual a A[0]) e o número de vezes que a operação básica é executada é 1.

B.(f) O melhor caso se dá quando o elemento procurado está na primeira posição da estrutura (p é igual a A[0]) e o número de vezes que a operação básica é executada é N.

C.(v) O pior caso acontece quando, determinada uma entrada, de tamanho n, a operação básica executada é a maior quantidade de vezes possível. Neste caso, o elemento procurado não existe na estrutura e são realizadas n comparações ao final da operação. Nesse caso, o algoritmo tem complexidade O(n).

D. (f) O pior caso acontece quando, determinada uma entrada, de tamanho n, a operação básica executada é a maior quantidade de vezes possível. Neste caso, o elemento procurado não existe na estrutura e são realizadas n comparações ao final da operação. Nesse caso, o algoritmo tem complexidade $O(n^2)$.

a.( ) A-F B-F C-V D-F

b.( ) A-V B-F C-F D-V

c.(x) A-V B-F C-V D-F

d.( ) NDA

5. Sobre o algoritmo abaixo: (1.0 pt)

```
int x(int v[],int n){
   int i = 0;
   while (i<n-1) {
        if (v[i]>v[i+1])
          return 0;
        i++;
   }
   return 1;}
```

a. (x) O algoritmo verifica se a estrutura v[] está em ordem crescente. No pior caso o algoritmo tem complexidade O(n), no melhor caso a complexidade de tempo é uma constante O(1).
b. ( ) O algoritmo verifica se a estrutura v[] está em ordem decrescente. No pior caso o algoritmo tem complexidade O(n), no melhor caso a complexidade de tempo é uma constante O(1).
c. ( ) O algoritmo verifica se a estrutura v[] está em ordem crescente. No pior caso o algoritmo tem complexidade $O(n^2)$, no melhor caso a complexidade de tempo é uma constante O(1).
d. ( ) nda

6. Sobre o código abaixo: (1.0 pt)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
class No{
        public:
                char nome;
                No *prox;
                No(char n){
                        nome=n;
                        prox=NULL;
                }
};
class X{
        public:
                No *inicio;
                No *fim;
```

```
            X(){
            inicio = NULL;
            fim = NULL;

            }
            void insereX(char n){
                  No *novo= new No(n);
                  if (inicio==NULL){
                        inicio = novo;
                        fim = novo;
                  }
                  else{
                        fim->prox=novo;
                        fim=novo;
                  }
            }
            char retiraX(){
                  No *el;
                  char nome;
                  if (inicio!=NULL){
                        el=inicio;
                        nome=el->nome;
                        inicio=inicio->prox;
                        free(el);
                  }
                  return nome;
            }
            int vaziaX(){
                  return (inicio==NULL);
            }
            void mostra(){
                  No *temp;
               cout<<"\n\n--------Mostra ----------\n\n";
               while (inicio!=NULL){
                  temp = inicio;
                  printf("\nNome %c\n",inicio->nome);
                        inicio = inicio->prox;
                        free(temp);
               }
         }
      };

main(){
   X *l1=new X();
   X *l2=new X();
   int resp;
   char letra;


   do {
         cout<<"\nDigite a letra:";
         cin>>letra;
```

```
        if (l1->vaziaX()==1)
           l1->insereX(letra);
        else{
                while (l1->vaziaX()!=1){
                   l2->insereX(l1->retiraX());
                }
                l1->insereX(letra);
                while (l2->vaziaX()!=1){
                   l1->insereX(l2->retiraX());
                }
         }
        cout<<"\nDeseja continuar (1-Sim 2-Nao)?";
        cin>>resp;
  }while(resp==1);
  l1->mostra();
}
```

Para uma entrada igual a:

Digite a letra:A
Deseja continuar (1-Sim 2-Nao)?1
Digite a letra:B
Deseja continuar (1-Sim 2-Nao)?1
Digite a letra:C
Deseja continuar (1-Sim 2-Nao)?1
Digite a letra:D
Deseja continuar (1-Sim 2-Nao)?2

A saída do programa será igual a:

a.(X) Nome D Nome C Nome B Nome A	b.( ) Nome A Nome B Nome C Nome D
c.( ) Não apresentará nada pois a estrutura “l1” está vazia ao final do programa.	d.( ) NDA

7. Considere “L” uma estrutura de lista simplismente encadeada. Sobre o código abaixo, responda:

```
int f(char x,Lista L){
   if (L == NULL) return 0;
   if (x == L->item) return 1;
   return f(x,L->prox);
}
```

7.1. Para uma L igual a [10,12,7] e para x igual a 3: (1.0 pt)
a.( ) A função f vai fazer a primeira chamada f(3,[10,12,7]) e mais 2 chamadas recursivas.
b.(x) A função f vai fazer a primeira chamada f(3,[10,12,7]) e mais 3 chamadas recursivas.
c.( ) A função f vai fazer a primeira chamada f(3,[10,12,7]) e mais 1 chamada recursiva.
d.( ) nda

7.2. Para uma L igual a [10,12,7] e para x igual a 10: (1.0 pt)
a.(x) A função f vai fazer a primeira chamada f(10,[10,12,7]), retornará 1 e não vai fazer nenhuma chamada recursiva.
b.( ) A função f vai fazer a primeira chamada f(10,[10,12,7]), retornará 1 e vai fazer 1 chamada recursiva.
c.( ) A função f vai fazer a primeira chamada f(10,[10,12,7]), retornará 0 e não vai fazer nenhuma chamada recursiva.
d.( ) A função f vai fazer a primeira chamada f(10,[10,12,7]), retornará 0 e vai fazer 1 chamada recursiva.