Prova – P2

Descrição
- Proponha solução para a prova abaixo, individualmente.

O que entregar
- Documento contendo nome, RA, enunciado da prova e as soluções propostas.

Quando entregar
- Até 2 horas após o início da prova.

Onde entregar
- No ambiente de interação da disciplina no Google Classroom, no link indicado.

Nome: Leticia Bossatto Marchezi
RA: 791003
Data: 25/06/2021

Orientações Gerais
- **Tempo para elaboração da prova: 2:00h (2 horas e 00 minutos**).

Orientações Quanto a Notação, Nomes das Variáveis, e Estruturas
- Use os **mesmos nomes fornecidos no enunciado**. Utilize variáveis auxiliares temporárias, o tanto quanto for necessário. É só declarar e usar. Mas **não considere a existência de nenhuma outra variável permanente ou funções**, salvo **se explicitamente indicado no enunciado da questão.**
- Considere as **estruturas exatamente conforme definido no enunciado**, seja no texto da questão, seja nos diagramas.
- Para o desenvolvimento de algoritmos, use preferencialmente a notação de C ou C++.

Questão 1 (4 pontos)

Considere que o nó de uma árvore binária de busca tenha a seguinte declaração:

```
typedef struct node {
        int chave;
        struct node *esq;
        struct node *dir;
} Node;
```

Observe que no nó acima a informação sobre o tamanho da árvore não está armazenada.

Considere que o tamanho de uma árvore R é o número de nós da árvore, ou seja:
Para R = NULL o tamanho da árvore é 0;
Para R != NULL o tamanho da árvore é composto pelo número de nós da sub-árvore esquerda de R + número de nós da sub-árvore direita de R + 1

a) Escreva uma função que dada uma árvore R, retorne o tamanho da mesma. A função deve ter a seguinte assinatura:

Linguagem C

```c
int tamDir, tamEsq;
    tamDir=0;
    tamEsq=0;

    // caso base, ponteiro é nulo
    if(R == NULL){
        return 0;
    }
```

```
    // calcula o tamanho de cada no filho
    tamEsq = getSize(R->esq);
    tamDir = getSize(R->dir);

    // retorna a soma
    return (1 + tamEsq + tamDir);

}
```

b) Qual é a ordem de eficiência de tempo a função do item (a).

A ordem de eficiência do tempo é O(n) onde n é quantidade de nós da árvore, pois a função percorre cada nó somente uma vez para calcular a quantidade de nós de seus filhos até atingir o caso base.

c) A função abaixo faz uma busca em uma Árvore Binária de Busca de raiz R. A função abaixo é ineficiente. Altere a função para que ela passe a ter a eficiência O(h), onde h é a altura da árvore binária.

```
// Retorna um ponteiro para o nó com a chave ch ou NULL
Node * busca (Node * R, int ch){
   if (R == NULL)
      return NULL;
   if (R ->chave == ch)
      return R;
   Node * resp1 = busca(R ->esq,ch);
   Node * resp2 = busca(R ->dir,ch);
   if(resp1!=NULL)
      return resp1;
   return resp2;
}
```

Linguagem C

```
Node * buscaOtimizado (Node * R, int ch){
    // Casos bases:

    // Ponteiro nulo retorna null
    if (R == NULL)
        return R;
    // Achou a chave
    if (R->chave == ch)
        return R;

    // Chamada recursiva
    // Se o no atual é maior que a chave visita o nó a esquerda
    if(R->chave > ch){
        return buscaOtimizado(R->esq, ch);
    }
    // Caso contrário visita o nó a direita
    else{
        return buscaOtimizado(R->dir, ch);
    }
```

```
}
```

Questão 2 (3 Pontos)

Escreva uma função que decida se um max-heap armazenado em um vetor v[0 .. m - 1] é ou não coerente. Para ser um max-heap coerente o filho esquerdo de um nó tem que ser sempre menor que o filho direito, ou seja, para um nó i, v[fesq(i)]<v[fdir(i)]. Use as definições abaixo na sua função.

```
#define pai(i) ((i – 1) / 2 )
#define fesq(i) (i * 2  +  1)
#define fdir(i) (i * 2  +  2 )
```

Linguagem C++

```
bool testaMaxHeap(int *vet, int num){
    int contador;
    contador=0;
    // laço para testar todos os valores
    while(fesq(contador)<fesq(contador)-1){
          // Se a chave do nó a esquerda não for menor do que o da direita, não é
coerente
        if(vet[fesq(contador)]>=vet[fdir(contador)]){
            // resultado nao é coerente
            return 0;
        }
        // atualiza o contador para o próximo nó
        contador=(fesq(contador)+fesq(contador))/2;
    }
    // resultado é coerente
    return 1;
}
```

Questão 3 (3 Pontos)

a) Escreva uma **função de ordenação** que ordena um vetor v de tamanho n.

Linguagem C

```
void ordena(int * v,int n){
    int i, j, auxiliar;
    j = 1;

    for(i = 0; i<n; i++){ // laço para percorrer todos os valores
        while(j < n - i){
              if(v[j-1] > v[j]){ // testa se o elemento anterior é maior do que o
prox
                // faz a troca dos valores
                auxiliar = v[j-1];
                v[j-1] = v[j];
                v[j] = auxiliar;
            }
            j++;
        }
    }
}
```

b) Qual é o custo computacional em número de operações da função implementada no item (a)?

O custo de tempo da função é O(n^2), sendo n o número de elementos do vetor, pois há laços aninhados em que cada um realiza n-1 operações, ou seja, no total são (n-1)^2, sendo possível simplificar para n^2.

c) Qual é o custo dessa função em termos de memória da função implementada no item (a)?

O custo em termos de armazenamento é de O(1), pois há a criação de apenas uma variável (auxiliar) para o devido funcionamento da função, independente da quantidade de elementos do vetor.

d) Responda: A **busca binária** é mais ou menos eficiente do que a busca sequencial? Por que? explique.

A busca binária tem complexidade O(lg n) enquanto a busca sequencial tem custo O(n), isso ocorre pois a busca binária não precisa percorrer todos os elementos em uma lista ordenada como a busca sequencial faz. Na verdade, a busca binária faz sucessivas comparações com o elemento central e divide a lista original em 2: a primeira com elementos menores do que o central, e a segunda sendo os maiores; assim, se o elemento buscado é diferente do central e é menor do que ele, o algoritmo se repete com a primeira parte da lista, caso o elemento seja maior do que o central testa a segunda parte da lista. Logo, há menos execuções necessárias ao realizar busca binária do que na busca sequencial, sendo então um algoritmo mais eficiente.