Aluno(a):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ matrícula:\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| 1a) | 2,0 |  |
| 2a) | 3,0 |  |
| 3a) | 3,0 |  |
| 4a) | 2,0 |  |
|  |  |  |

* A prova é individual e sem consulta.
  + **Qualquer tentativa de “cola” resultará na anulação da prova do aluno ou dos alunos envolvidos**.
  + Os aparelhos celulares deverão permanecer desligados durante toda a prova. **Aparelhos celulares ligados serão tratados como tentativa de “cola”**.
* A interpretação faz parte da questão.
  + **Não há perguntas durante a prova.**
  + Em caso de dúvida escreva a dúvida e a sua interpretação na resposta.
* O tempo de prova é 1:45 h.
* **Após o início da prova, não será possível sair e depois voltar à sala.**
* As respostas devem seguir as questões. Caso precise de rascunho use o verso da folha.
* A prova pode ser feita a lápis.

**Questão 1** (2,0 pontos) Os algoritmos abaixo são usados para resolver problemas de tamanho *n*. Determine a complexidade, no pior caso, de cada algoritmo. Explique sua resposta.

a) (1,0 ponto)

int funcao\_C(int n) {

int i, j, k, sum = 0;

for ( i=1; i < n; i \*= 2 ) {

for ( j = n; j > 0; j -= 2 ) {

for ( k = j; k < n; k += 2 ) {

sum += (-j \* k) << i/2;

}

}

}

return sum;

}

b) (1,0 ponto)

int funcao\_C(int n) {

int i, j, sum = 0;

for (i=0; i<n; i++) {

for (j=0; j<i; j++) {

sum+=j;

}

}

for (j=0; j<n; j++){

sum+=2\*j;

}

return sum;

}

**Resposta**

a) O laço mais externo é executado *log2(n)* vezes, visto que *i* dobra a cada passagem. O laço do meio e o último laço são executados *n/2* vezes. Como os laços estão aninhados, o comando “sum += (-j \* k) << i/2” é executado *(n/2)2log2(n)* vezes. Logo, a complexidade desse algoritmo é *O(n2(log n))*.

b) Os dois laços iniciais estão aninhados e cada um é executado *n* vezes. Logo, o comando “sum+=j” é executado *n2* vezes. O terceiro laço é executado *n* vezes. Como este laço está em paralelo com os laços aninhados, a complexidade deve ser somada, o que resulta em *O(n2+n)=O(n2)*.

**Questão 2** (3,0 pontos) Uma tabela de dispersão de tamanho 11 é implementada com encadeamento externo (resolução de colisões por listas encadeadas) através da seguinte função de dispersão:

/\* resto da divisão de x por 11 \*/

* 1. (1,5 ponto) Desenhe a estrutura de dados após a inserção das chaves (nesta ordem):

7,10,15,14,17,3,21,25

* 1. (0,5 ponto) Apresente uma sequencia de 11 chaves para as quais a função de hash produz um número máximo de colisões. Explique a sua resposta.
  2. (1,0 ponto) Considere uma tabela de hash de tamanho fixo *n*. Suponha que *m* chaves tenham sido inseridas na tabela, onde *m* < *n*. Qual o custo, no pior caso, da pesquisa por uma chave? Explique a sua resposta.

**Resposta**

a)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *x* | *h(x)* |  |  |  |  |
| 7 | 7 |
| 10 | 10 |
| 15 | 4 |
| 14 | 3 |
| 17 | 6 |
| 3 | 3 |
| 21 | 10 |
| 25 | 3 |

|  |  |
| --- | --- |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |  | 14 |  | 3 |  | 25 |
| 4 |  |  | 15 |
| 5 |  |  |  |
| 6 |  |  | 17 |
| 7 |  |  | 7 |
| 8 |  |
| 9 |  |
| 10 |  |  | 10 |  | 21 |

b) Basta inserir chaves cujos restos da divisão por 11 sejam iguais, por exemplo, *C*={11, 22, 33, 44, 55, 66, 77, 88, 99, 110, 121}. De fato, *hash(x)=*0 para todo x ∈ *C*. Portanto, todas estas chaves serão inseridas na mesma lista encadeada associada à entrada 0 da tabela.

c) No pior caso, a inserção de *m* chaves em uma tabela de hash de tamanho *n*, onde *m* < *n*, provoca *m* colisões. Ou seja, todas as chaves serão inseridas na mesma lista. Portanto, o custo de uma pesquisa após *m* inserções será proporcional a *m*, ou seja, *O(m)*.

**Questão 3** (3,0 pontos) Considere conjuntos de elementos com suas prioridades. Assuma que as prioridades são números inteiros e que os conjuntos são representados apenas pelas prioridades dos seus elementos, como nos exemplos em sala.

* 1. (1,5 ponto) Considere o conjunto *C*={90, 60, 30, 15, 45}. Mostre, passo a passo, como um vetor armazenando um *heap* mínimo para o conjunto *C* é construído, da melhor maneira possível. Comente brevemente cada passo do algoritmo.
  2. (1,5 ponto) Mostre, passo a passo, como inserir 5 no vetor abaixo, que armazena um *heap* mínimo. Comente brevemente cada passo da inserção.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| posição | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  | … |
| prioridade | 7 | 9 | 18 | 15 | 17 | 21 | - |  | … |

**Resposta**

a)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |  |
|  |  | 90 | 60 | 30 | coloque 3 elementos no final do vetor |
|  | 15 | 90 | 60 | 30 | coloque 15 na primeira posição não ocupada;  o filho à esquerda de 15 é 60 e o filho à direita é 30  como 15 é menor, não há trocas | | | | |  |
| 45 | 15 | 90 | 60 | 30 | coloque 45 na primeira posição não ocupada;  o filho à esquerda de 45 é 15 e o filho à direita é 90 | |  | |
| 15 | 45 | 90 | 60 | 30 | troque 45 com 15  o filho à esquerda de 45 é 60 e o filho à direita é 30 | |  |
| 15 | 30 | 90 | 60 | 45 | troque 45 com 30; pare | |  |

b)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |  | … |  |
| 7 | 9 | 18 | 15 | 17 | 21 | - |  | … |  |
| 7 | 9 | 18 | 15 | 17 | 21 | 5 | - |  | coloque 5 na última posição  compare 5 com o pai, na posição (6-1)/2=2 |
| 7 | 9 | 5 | 15 | 17 | 21 | 18 | - |  | como 5 é menor, troque 5 com 18  compare 5 com o pai, na posição (2-1)/2=0 |
| 5 | 9 | 7 | 15 | 17 | 21 | 18 | - |  | como 5 é menor, troque 5 com 7  como 5 está no primeiro elemento do vetor, pare |

**Questão 4** (2,0 pontos)

Considere a representação de conjuntos utilizando vetores de bits, definidos como:

struct \_bitvector {

int max; /\* número máximo de elementos do conjunto \*/

int \*vector;

};

typedef struct \_bitvector BitVector;

a) (1,5 ponto) Implemente em C uma rotina que recebe como entrada dois conjuntos *A* e *B* e devolve o conjunto *D* = *A* – *B*. A rotina não deverá chamar as operações definidas em sala para vetores de bits.

Nota: *A* – *B*, a *diferença* de *A* e *B*, é o conjunto de todos os elementos de *A* que não estão em *B*.

b) (0,5 ponto) Qual a complexidade, no pior caso, da rotina implementada no item (a), medida em termos do número de elementos dos conjuntos. As operações bit-a-bit influenciam a complexidade, no pior caso? Explique cuidadosamente a sua resposta.

**Resposta**

**a)** BitVector\* bvN(BitVector\* A, BitVector\* B)

{

int i;

int minab = (A->max < B->max) ? A->max : B->max;

int minnum = (minab-1) /sizeof(int)+1; /\* tamanho do menor \*/

int num = ((A->max-1)/sizeof(int))+1; /\* tamanho do vetor D \*/

BitVector\* D = (BitVector\*)malloc(sizeof(BitVector));

D->max = A->max; /\* D terá o número de elementos de A \*/

D->vector = (int\*)malloc(num\*sizeof(int));

for (i=0; i<minnum; i++)

D->vector[i] = A->vector[i] & ~(B->vector[i]);

for (i=minnum; i<num; i++) /\* copia o resto de A para D, \*/

D->vector[i] = A->vector[i]; /\* se preciso \*/

return D;

}

(b) Em função dos for-loops, a rotina possui complexidade *O(n)*, onde *n* é o número de elementos do conjunto *A*. Note que as operações bit-a-bit não influenciam a complexidade, no pior caso, pois apenas dividem *n* por *b*, o número de bits necessários para representar cada elemento do vetor. Como *b* é uma constante, é absorvido pelo notação *O*, ou seja, *O(n)* = *O(n/b)*, quando *b* é uma constante.