UENF

Universidade Estadual do Norte Fluminense Darcy Ribeiro

Curso: Ciência de Computação **Data:** 12./.09./2022
Atividade: Questões Prova 1 **Período:** 4º **Disciplina:** Estrutura de dados II
Professor: Fermín Alfredo Tang **Turno:** Diurno
Nome do Aluno: .....Matrícula: .....

1. Responda com verdadeiro (V) ou falso (F), justificando a sua resposta:

   i) Sempre é possível afirmar que um algoritmo A é melhor que outro Algoritmo B apenas comparando seus tempos de execução para um tamanho fixo de entrada n.

   ii) A função $T(n)$ mede de forma precisa o número de operações realizadas por um algoritmo em função da entrada do algoritmo.

   iii) A função $E(n)$ mede de forma aproximada o número de posições de mémória realizadas por um algoritmo em função da entrada do algoritmo.

   iv) A notação $O(f(n))$ serve para classificar o comportamento de um algoritmo com função $T(n)$ na categoria $f(n)$ mas não para medir seu desempenho.

   v) Na notação $O(f(n))$ podemos descartar termos de menor grau e coeficientes multiplicativos, o que significa que eles não são necessários para medir o seu desempenho.

   Respostas

   i) Falso. Esse tipo de comparação pontual não permite concluir se um algoritmo A é melhor que outro algoritmo B.

   Em geral, podemos ter as seguintes situações:

   i) Um algoritmo é sempre melhor que o outro. Por exemplo: a função de tempo $T_A(n)$ é sempre menor que $T_B(n)$.

   ii) Um algoritmo é melhor que o outro somente a partir de um valor de n suficientemente grande. Por exemplo: $T_A(n) < T_B(n)$ para $n \geq n_0$.

   ii) Falso. A função $T(n)$ mede o número de operações realizadas por um algoritmo em função da entrada do algoritmo. Mas essa medição não deixa de ser uma aproximação da realidade. Já que a contagem de operações sempre sofre simplificações.

   iii) Verdadeiro. Usando a explicação dada acima aplicada ao caso da memória.

   iv) Verdadeiro. A função $f(n)$ dá uma ideia geral da taxa de crescimento do algoritmo (em tempo ou memória) e pode servir como limite assintótico. Mas deixa de fora constantes multiplicativas e termos de menor grau que são necessários para medir o desempenho.

   v) Falso. Pelo explicado anteriormente, todos os termos da função $T(n)$ são necessários para medir o desempenho do algoritmo (em tempo ou memória).

2. Considerando o seguinte vetor:

0	1	2	3	4	5	6	7	8	9
15	5	16	4	10	23	18	39	26	2

Execute o **algoritmo shellsort** considerando espaçamento $h = 4$ e obtenha a ordenação em 4. Observe que o algoritmo começa tentando inserir o elemento que se encontra na posição 4, no seu subconjunto correspondente. E prosegue tentando inserir os elementos das posições 5 até 9.

Indique o número de comparações.
Indique o número de deslocamentos.

Resposta2.-

i) Execute o **algoritmo shellsort** considerando espaçamento $h = 4$ e obtenha a ordenação em 4. Observe que o algoritmo começa tentando inserir o elemento que se encontra na posição 4, no seu subconjunto correspondente. E prosegue tentando inserir os elementos das posições 5 até 9.

0	1	2	3	4	5	6	7	8	9
15	5	16	4	10	23	18	39	26	2
10	5	16	4	15	23	18	39	26	2
10	5	16	4	15	23	18	39	26	2
10	5	16	4	15	23	18	39	26	2
10	5	16	4	15	23	18	39	26	2
10	5	16	4	15	23	18	39	26	2
10	2	16	4	15	5	18	39	26	23

Vetor ordenado $h = 4$

Indique o número de comparações:

1+1+1+1+1+2 = 7
Indique o número de deslocamentos: 1+0+0+0+0+2 = 3

3. Determine o valor do espaçamento inicial h, no algoritmo de Shellsort, usando a função de Knuth, considerando um vetor de tamanho n igual a:

   i) 500 elementos;

   ii) 5.000 elementos;

   iii) 50.000 elementos;

   Em cada caso, usando a função inversa de Knuth, indique também a sequência de valores de h usada na redução do espaçamento até $h = 1$

   Resposta3.-

   Usando a função de Knuth: $3h + 1$, para valores menores do que n e $(n - 1)/3$, temos respectivamente:

h	$3h+1$	
1	4	
4	13	
13	40	
40	121	$(n-1)/3 = 166$
121	364	$n = 500$
364	1.093	$(n-1)/3 = 1.666$
1.093	3.280	$n = 5.000$
3.280	9.841	$(n-1)/3 = 16.666$
9.841	29.524	$n = 50.000$
29.524	88.573	

i) 500 elementos; $h = 121$

Usando a função inversa de Knuth temos a sequência:

40, 13, 4, 1.

ii) 5.000 elementos; $h = 1.093$

Usando a função inversa de Knuth temos a sequência:

364, 121, 40, 13, 4, 1.

n	$(n-1)/3$
500	166
5.000	1.666
50.000	16.666

3.280,

iii) 50.000 elementos; $h = 9.841$

Usando a função inversa de Knuth temos a sequência:

1.093, 364, 121, 40, 13, 4, 1.

4. Considerando o seguinte vetor:

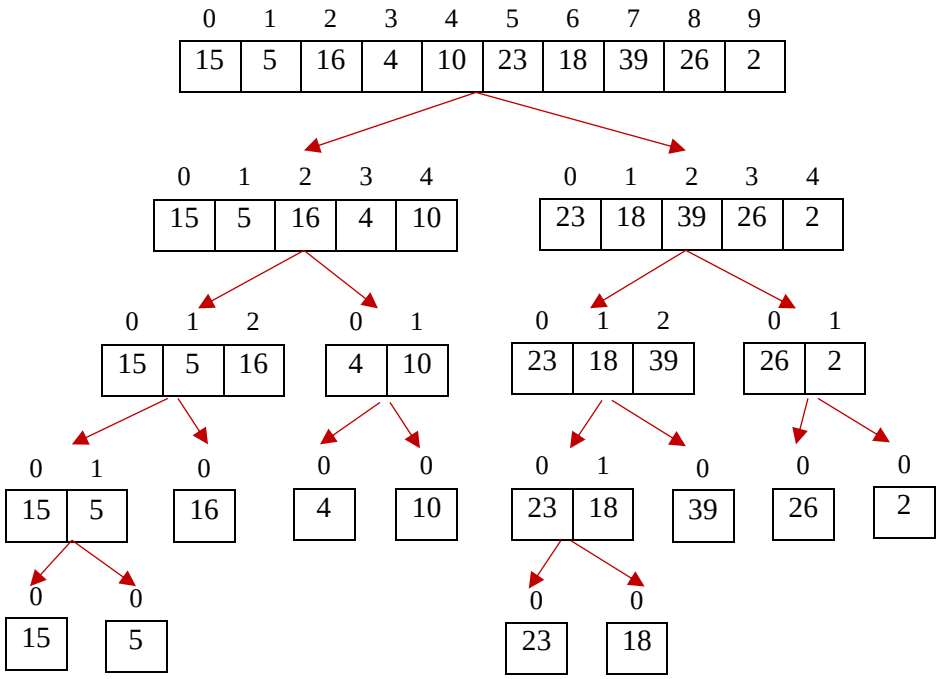
0	1	2	3	4	5	6	7	8	9
15	5	16	4	10	23	18	39	26	2

Execute o **algoritmo mergesort** mostrando apenas as árvores de: i) decomposição em subproblemas mediante copias e ii) composição dos subproblemas ordenados mediante copias.

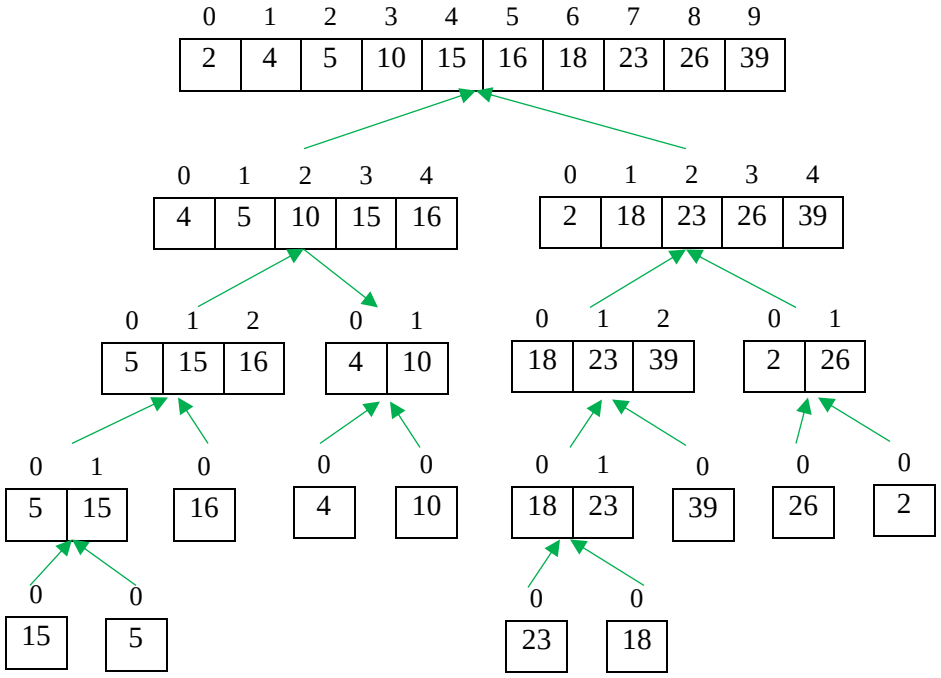
Indique o número de comparações.
Indique o número de copias.

Resposta 4.-

i) A árvore de decomposição em subproblemas é:



ii) A árvore de composição de subproblemas ordenados é:



Indique o número de comparações: Subida: Nivel 4: 2
Nivel 3: 2 + 1 + 2 + 1 = 6
Nivel 2: 3 + 4 = 7
Nivel 1: 6
Total: 21

Indique o número de copias: Descida: 3(10) + 4
Subida: 3(10) + 4

Total: 68 copias

5. Considerando os seguintes vetores:

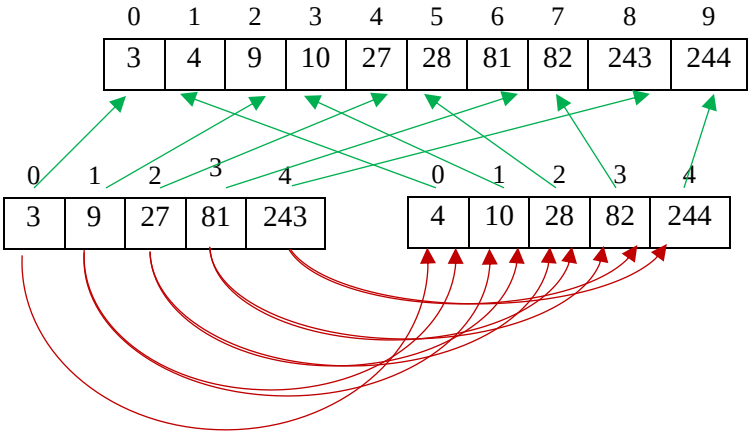
0	1	2	3	4
3	9	27	81	243

0	1	2	3	4
4	10	28	82	244

Execute o **algoritmo merge** de intercalação. Indique o número de comparações.

Generalizando esta situação para dois vetores de tamanho $n/2$. Qual seria o número de comparações em termos de n?. Isso representa o melhor caso ou pior caso?.

Resposta6.- Como os elementos do primeiro e segundo vetor se encontram perfeitamente intercalados o menor elemento será sempre alternado entre um elemento do primeiro vetor e outro do segundo vetor. Dessa forma, ambos vetores diminuem de tamanho juntos, sendo realizadas 9 comparações até esgotar um deles. O menor elemento é copiado no vetor ordenado com 10 elementos.



Generalizado a situação para dois vetores de tamanho $n/2$, o número de comparações seria $n-1$. Isso representa o pior caso.

6. Considerando o seguinte vetor:

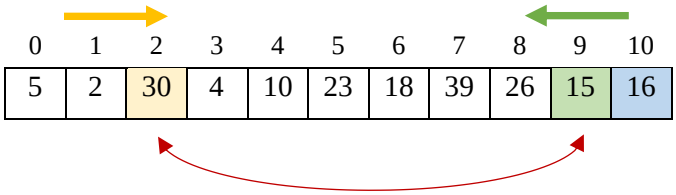
0	1	2	3	4	5	6	7	8	9	10
5	2	30	4	10	23	18	39	26	15	16

Execute o **algoritmo particiona** do **algoritmo quicksort**

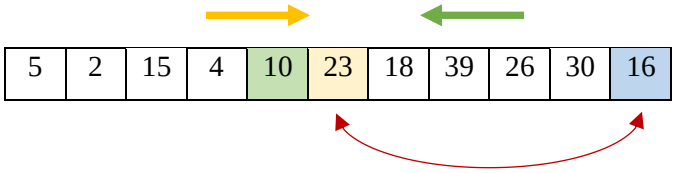
Indique o número de comparações.
Indique o número de trocas.

Resposta6.- Usamos como elemento pivô o ultimo elemento. Usamos a versão 2, do Quicksort que percorre os elementos do vetor simultaneamente pela esquerda e pela direita.

O algoritmo percorre os elementos pela esquerda enquanto forem menores que 16. Ao encontrar o elemento 30 para. Também percorre os elementos pela direita enquanto forem maiores que 16. Ao encontrar o elemento 15 para. Os elementos 15 e 30 são trocados de posição, porque estão no lado errado.



O algoritmo retoma o percursos pela esquerda e pela direita. Ambos percursos param em 23 e 10 respectivamente. Mas desta vez a posição de parada do percurso esquerdo cruzou com a posição de parada do percurso direito. O que significa que não há elementos fora de lugar.



Finalmente, o pivô é trocado com a posição de parada esquerda.

5	2	15	4	10	16	18	39	26	30	23

Indique o número de comparações: 12
Indique o número de trocas: 2

7. Considere a sequência de chaves mostrada abaixo, processadas linha por linha, e uma tabela hash com 19 elementos.

```
224562 137456 214562
140145 214576 162145
144467 199645 234534
```

Use o método do módulo da divisão para *hashing* inicial e endereçamento aberto com tentativa linear para resolver colisões;

Resposta7.- Sabemos que a tabela possui 19 elementos, com posições de 0 a 18.

Usando o módulo da divisão temos: $h(x) = x \; mod \; 19$

Na tentativa linear fazemos: $h(x, i) = (h(x) + i) \; mod \; 19$

Assim:

1) Para $x = 224562$ temos: $h(224562) = 224562 \; mod \; 19 = 1$
2) Para $x = 137456$ temos: $h(137456) = 137456 \; mod \; 19 = 10$
3) Para $x = 214562$ temos: $h(214562) = 214562 \; mod \; 19 = 14$
4) Para $x = 140145$ temos: $h(140145) = 140145 \; mod \; 19 = 1$

   Existe colisão, para $i = 1$ temos: $h(140145, 1) = \; (1 + 1) \; mod \; 19 = 2$
5) Para $x = 214576$ temos: $h(214576) = 214576 \; mod \; 19 = 9$
6) Para $x = 162145$ temos: $h(162145) = 162145 \; mod \; 19 = 18$
7) Para $x = 144467$ temos: $h(144467) = 144467 \; mod \; 19 = 10$

   Existe colisão, para $i = 1$ temos: $h(144467, 1) = \; (10 + 1) \; mod \; 19 = 11$
8) Para $x = 199645$ temos: $h(199645) = 199645 \; mod \; 19 = 12$
9) Para $x = 234534$ temos: $h(234534) = 234534 \; mod \; 19 = 17$

A tabela *hash* é preenchida da seguinte forma:

0	
1	224562
2	140145
3	
4	
5	
6	
7	
8	
9	214576
10	137456
11	144467
12	199645
13	
14	214562
15	
16	
17	234534
18	162145

8.- Na mesma sequência da questão 7, use o método de *hashing* da extração de dígitos (primeiro, terceiro e quinto dígitos), lembre de normalizar os endereços usando o módulo da divisão, caso necessário. Use o endereçamento aberto com tentativa quadrática para o tratamento das colisões;

Resposta2.- Fazendo a extração de três dígitos ainda teremos chaves com valores maiores que a dimensão da tabela, com isso teremos que normalizar usando o módulo da divisão: $h(x) = x \; mod \; 19$

Na tentativa quadrática utilizamos como novo endereço o lugar da colisão mais o quadrado da tentativa. Caso necessário, normalizar.

Assim:

1) Para $x = 224562$ extraímos: $h(246) = 246 \; mod \; 19 = 18$
2) Para $x = 137456$ extraímos: $h(175) = 175 \; mod \; 19 = 4$
3) Para $x = 214562$ extraímos: $h(246) = 246 \; mod \; 19 = 18$

   A chave simplificada é idêntica a outra chave anterior. Existe colisão em 18.

   Para a tentativa $i = 1$ a nova posição seria: $18 + (1)^2 = 19$

Aplicando módulo da divisão temos: $h(x) = 19\ mod\ 19 = 0$

4) Para $x = 140145$ extraímos: $h(104) = 104\ mod\ 19 = 9$
5) Para $x = 214576$ extraímos: $h(247) = 247\ mod\ 19 = 0$

   Existe colisão em 0.

   Para a tentativa $i = 1$ a nova posição seria: $0 + (1)^2 = 1$

6) Para $x = 162145$ extraímos: $h(124) = 124\ mod\ 19 = 10$
7) Para $x = 144467$ extraímos: $h(146) = 146\ mod\ 19 = 13$
8) Para $x = 199645$ extraímos: $h(194) = 194\ mod\ 19 = 4$

   Existe colisão em 4.

   Para a tentativa $i = 1$ a nova posição seria: $4+(1)^2 = 5$

9) Para $x = 234534$ extraímos: $h(243) = 243\ mod\ 19 = 15$

A tabela *hash* é preenchida da seguinte forma:

0	214562
1	214576
2	
3	
4	137456
5	199645
6	
7	
8	
9	140145
10	162145
11	
12	
13	144467
14	
15	234534
16	
17	
18	224562

9.- Na mesma sequência da questão 7, use o método midsquare, com os dois dígitos centrais para *hashing* inicial, (lembre de normalizar os endereços usando o módulo da divisão, caso necessário). Use o método key-offset para tratamento de colisões.

Resposta9.-

O método midsquare usará o quadrado dos dois dígitos centrais usando o módulo da divisão: $h(x) = x \; mod \; 19$ para normalizar o resultado.

Em caso de colisão, o método *key offset* deve utilizar a seguinte expressão para calcular o deslocamento: $offset = \lfloor x/19 \rfloor$, onde x representa o quadrado.

A nova chave será calculada somando o deslocamento à posição colidida e normalizando.

$$novo\ endereço = (colisão + offset)\ mod\ 19$$

Assim:

1) Para $x = 224562$ extraímos 45: $h(45^2) = 2025\ mod\ 19 = 11$
2) Para $x = 137456$ extraímos 74: $h(74^2) = 5476\ mod\ 19 = 4$
3) Para $x = 214562$ extraímos 45: $h(45^2) = 2025\ mod\ 19 = 11$

   A chave simplificada é idêntica a outra chave anterior. Existe colisão em 11.

   Para a tentativa $i = 1$, calculamos o $offset = \lfloor 2025/19 \rfloor = 106$

   $$novo\ endereço = (11 + 106) mod\ 19 = 3$$
4) Para $x = 140145$ extraímos 01: $h(1^2) = 1\ mod\ 19 = 1$
5) Para $x = 214576$ extraímos 45: $h(45^2) = 2025\ mod\ 19 = 11$

   Existe colisão em 11. Calculamos o $offset = \lfloor 2025/19 \rfloor = 106$

   Para a tentativa $i = 1$, $novo\ endereço = (11 + 106)\ mod\ 19 = 3$

   Existe colisão em 3.

   Para a tentativa $i = 2$, $novo\ endereço = (3 + 106)\ mod\ 19 = 14$
6) Para $x = 162145$ extraímos 21: $h(21^2) = 441\ mod\ 19 = 4$

   Existe colisão em 4. Calculamos o $offset = \lfloor 441/19 \rfloor = 23$

   Para a tentativa $i = 1$, $novo\ endereço = (4 + 23)\ mod\ 19 = 8$
7) Para $x = 144467$ extraímos 44: $h(44^2) = 1936\ mod\ 19 = 17$
8) Para $x = 199645$ extraímos 96: $h(96^2) = 9216\ mod\ 19 = 1$

   Existe colisão em 1. Calculamos o $offset = \lfloor 9216/19 \rfloor = 485$

   Para a tentativa $i = 1$, $novo\ endereço = (1 + 485)\ mod\ 19 = 11$

   Para a tentativa $i = 2$, $novo\ endereço = (11 + 485)\ mod\ 19 = 2$
9) Para $x = 234534$ extraímos 45: $h(45^2) = 2025\ mod\ 19 = 11$

   Existe colisão em 11. Calculamos o $offset = \lfloor 2025/19 \rfloor = 106$

Para a tentativa $i = 1$, $novo\ endereço = (11 + 106)\ mod\ 19 = 3$

Existe colisão em 3.

Para a tentativa $i = 2$, $novo\ endereço = (3 + 106)\ mod\ 19 = 14$

Para a tentativa $i = 3$, $novo\ endereço = (14 + 106)\ mod\ 19 = 6$

A tabela *hash* é preenchida da seguinte forma:

0	
1	140145
2	199645
3	214562
4	137456
5	
6	234534
7	
8	162145
9	
10	
11	224562
12	
13	
14	214576
15	
16	
17	144467
18	

10.- Considerando questões sobre *hashing*, responda com verdadeiro ou falso as seguintes questões justificando em cada caso.

i) As técnicas de *hashing* garantem que o tempo de busca, inserção e remoção de um dado seja sempre O(1).

ii) A menos que o *hashing* seja perfeito independente da função de *hash* utilizada sempre existe a possibilidade de acontecer uma colisão.

iii) Podemos usar uma função de *hash* para o posicionamento inicial da chave na tabela e outra função de *hash* para o tratamento das colisões.

iv) Todo *Hash* duplo elimina o clustering secundário.

Resposta10.-

i) Falso. Embora o comportamento O(1) seja o desejado. Na prática, não existe garantia de que esse comportamento seja atingido, devido a ocorrência de colisões. Quanto maior for o fator de carga da tabela, maior será número de tentativas necessárias para encontrar um elemento.

ii) Verdadeiro. Somente quando o *hashing* é perfeito temos um mapeamento um a um entre as chaves e as posições na tabela *hash*. Caso contrário, apenas pelo fato de existirem um número muito maior e chaves potenciais do que posições disponíveis na tabela *hash* se torna inevitável a ocorrência de colisões.

iii) Verdadeiro. Essa abordagem é chamada de *hash* duplo.

iv) Falso. Nem todo *hash* duplo elimina o *clustering* secundário. O método de *key offset* elimina o *clustering* secundário. Já o *hashing* duplo pseudoaleatório não elimina o *clustering* secundário.