Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática (ICEI)
Curso de Ciência da Computação
Disciplina: Algoritmos e Estruturas de Dados II

Prova I

Aluno: Raquel de Parde Motta

1. Considere o código abaixo. Determine os itens que se seguem:

```
1  int x = 0;
2  for(int i = 15; i < n; i++){
3      for(int j = 0; j < 3; j++) x += a * 2;
4      for(int k = i; k < n; k++) {
5          x += b * 5;
6          x += x * 10;
7      }
8      x += c * 4
9  }
```

(a) a função de complexidade usando a notação $\sum$ para o número de multiplicações;

(b) a fórmula fechada da função de complexidadade e a ordem de complexidade usando a notação Θ.

2. Encontre a fórmula fechada para o somatório a seguir e, em seguida, faça a prova por indução matemática.

$$\sum_{i=1}^{n} [(10i+2)^2 - (10i+1)^2]$$

3. Considere a classe em Java denominada FilaPrioridade, que representa uma estrutura de dados do tipo Fila de Prioridade e que utiliza o conceito de máxima prioridade para determinar a ordem de remoção dos elementos. Elementos de maior prioridade devem ser removidos antes dos elementos de menor prioridade. Essa fila será utilizada para o atendimento de pacientes médicos, onde os pacientes são representados por nome e prioridade de atendimento..

```
1  class Paciente {
2      public String nome;
3      public int prioridade;
4  }
5  class FilaPrioridade {
6      private Paciente[] itens;
7      private int n;
8      public FilaPrioridade(int tam) { //IMPLEMENTAR }
9      public void inserir(Paciente p) { //IMPLEMENTAR }
10     public Paciente remover() { //IMPLEMENTAR }
11 }
```

Implemente o construtor e os métodos inserir e remover da fila. Considerações: o construtor gera uma fila que é capaz de armazenar, no máximo, "tam"pacientes; a inserção insere um paciente na fila, respeitando a ordem de prioridade; a remoção remove e retorna o paciente de maior prioridade da fila e, caso exista mais de um paciente com a mesma prioridade, remove o que foi inserido primeiro; os casos de estouro e underflow (remoção em fila vazia) devem ser detectados.

4. Modifique o algoritmo de ordenação Seleção para que, a cada iteração, seja selecionado tanto o menor quanto o maior elemento da porção não ordenada do arranjo. O menor elemento será movido para o início e o maior elemento será movido para o final da lista. Ao final do processo, o array estará ordenado de forma crescente. Os seguintes itens devem ser apresentados:

(a) Apresente o código em C-Like para essa versão modificada do Seleção.

(b) Explique como essa modificação impacta a complexidade temporal do algoritmo, apresentando a nova complexidade.

(c) Considere o array [29, 10, 14, 37, 13, 5, 26]. Mostre o estado do array após cada iteração usando o algoritmo modificado.

$$\sum_{i=15}^{n-1} 4 + 2n - 2i$$

15 16 17 18
↓ ↓ ↓ ↓
0+15 1+15 2+15 3+15

(b) $S_n = \underbrace{\sum_{15}^{n-1} 4}_{4(n-1-15+1)} + \underbrace{2 \cdot \sum_{15}^{n-1} n}_{2 \cdot n(n-1-15+1)} - 2 \cdot \sum_{15}^{n-1} i$

$4(n-1-15+1)$
$4.(n-15)$
$4n-60$

$2 \cdot n(n-1-15+1)$
$2n(n-15)$
$2(n^2-15n)$
$2n^2-30n$

mesma ideia

$2 \cdot \sum_{0}^{n-1} i+15$ → P.A.

$2\left(\underbrace{\sum_{0}^{n-1} i}_{\frac{(n-1)(n-1+1)}{2}} + \underbrace{\sum_{0}^{n-1} 15}_{15(n-1-0+1)=15n}\right)$ → GAUSS

$= \frac{(n-1).n}{2} = \frac{n^2-n}{2}$

montando a equação final:

$S_n = 4n-60+2n^2-30n-2.\left(\frac{n^2-n}{2}+15n\right)$

$S_n = 4n-60+2n^2-30n-\cancel{2}\left(\frac{n^2-n+30n}{\cancel{2}}\right)$

$S_n = 4n-60+2n^2-30n-n^2+n-30n$

$S_n = n^2-55n-60$ //

$\Theta(n^2)$

ordem de complexidade quadrática

Sei que algo deu errado mas não sei o que é... O 1º passo da prova por indução dona errado

---

(a)

manipulando o termo que está sendo somado:

$(10i+2)^2-(10i+1)^2 \rightarrow (a+b)^2 = a^2+2ab+b^2$

$= 100i^2+40i+4-(100i^2+20i+1)$

$= \cancel{100i^2}+40i+4\cancel{-100i^2}-20i-1$

$= 20i+3$

Portanto, temos: $\sum^{n} (20i+3)$

$$S_n = \sum_{1}^{n} 20i + \sum_{1}^{n} 2 = 20\underbrace{\sum_{1}^{n} i}_{\frac{n(n+1)}{2}} + \underbrace{\sum_{1}^{n} 3}_{3.(n-1+1)=3n}$$

$$S_n = 20\left(\frac{n^2+n}{2}\right) + 3n = 10(n^2+n) + 3n = 10n^2 + 10n + 3n$$

$$\boxed{S_n = 10n^2 + 13n}$$

Passo base da indução:

$10.(1)^2 + 13.1 = 23$ OK!

$(10.1+2)^2 - (10.1+1)^2 = (12)^2 - (11)^2 = 144 - 121 = 23$

Segundo passo:

$S_n = S_{n-1} + a_n$ (enésimo termo)

$S_n = 10(n-1)^2 + 13(n-1) + 20n + 3$

$S_n = 10(n^2 - 2n + 1) + 13n - 13 + 20n + 3$

$S_n = 10n^2 - 20n + 10 + 13n - 13 + 20n + 3$

$S_n = 10n^2 + 13n$ OK!

---

3)

```
// construtor
public FilaPrioridade (int tam) {
    itens = new Paciente [tam];
    n = 0;
}
```

P1 - Raquel de Parde Motta

```
void inserir (Paciente p){
    if (n == itens.length) {
        System.out.println("ERRO");
    }else{
        itens[n] = p;
        n++;
    }
}
```

// não dá p/ add pois o vetor já está cheio. Esqueci a sintaxe de exception em java então bloqueei p/ printar a msg de erro.

// o else garante que só prossegue se tiver espaço no vetor, evitando o estouro

inseri com prioridade

```
public Paciente remover ( ) {
    if (itens.length==0){
        System.out.println("ERRO");
    }else{
        Paciente temp = itens[0];
        for (int i=1 ; i < itens.length-1; i++) {
            itens[i-1] = itens[i];
        }
        n=n-1; return temp;
    }
}
```

// Precisamos garantir que a fila n esteja vazia para remover. Evita underflow

// arredonda os itens para trás. Vai sobrescrever a pessoa na posição 0, mas guardo ela em temp.

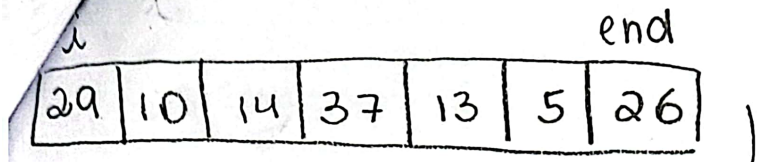
// fazemos n=n-1 para remover logicamente o item duplicado na última posição e encurtar de fato o vetor.

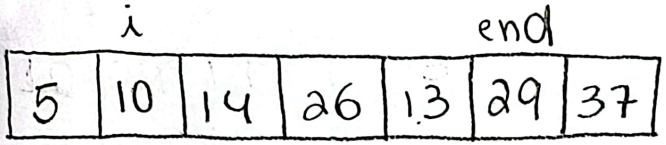
④
```
int [] selection (int[]arr, int n){   // tamanho (nº de elem. do array)
  for(int i=0; i< n-1; i++){
    int menor = arr[i];
    int maior = arr[i];      int end = n-1;
    for( int j = i+1; j<end, j++){
      if(arr[j] > maior) maior = arr[j];
      if(arr[j]< menor) menor = arr[j];
    }
    if(i >= end)
      i = n; //está ordenado. condição p/ interromper o for externo.
    }else{
      swap(arr, menor, i); //joga o menor p/ início
      swap(arr, maior, end); // e o maior p/ final
      end --;
    }
  }
  return arr;
}
```

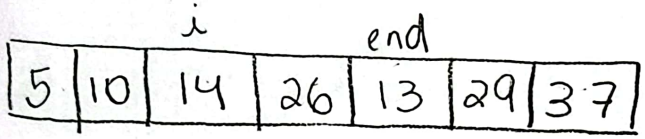
C+

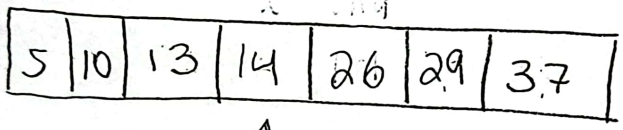
(b) Agora, procurando o menor e o maior elemento ao mesmo tempo, o array ficará ordenado antes e poderemos parar o for externo, que rodará metade das vezes que rodava antes. Portanto, a complexidade $\Theta(n^2)$ do selection original será reduzida a uma complexidade de tempo linear $\Theta(n)$.

P1 - Raquel de Parde Motta









O 13 vai pro lugar do 14

e o 14 vai pro lugar do 26

i
end

i = end para o for externo.