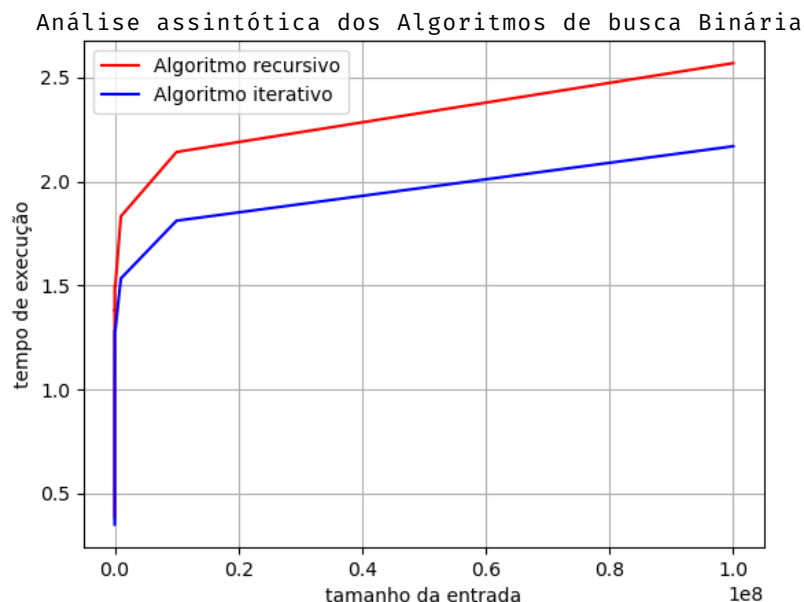


11. Escreva um algoritmo (em pseudocódigo) que realize busca binária de forma iterativa e o implemente numa linguagem de programação a sua escolha. Construa um gráfico mostrando a relação valor de entrada x tempo de execução do algoritmo implementado. Considerando uma análise assintótica em pior caso, explique se o desempenho do algoritmo implementado é superior, inferior ou igual ao do algoritmo que implementa busca binária de forma recursiva.

```
BINARY-SEARCH(listaOrdenada, valorProcurado, menor, maior):
01.     indiceDoElementoDoMeio = (menor + maior) // 2
02.     elementoDoMeio = listaOrdenada[indiceDoElementoDoMeio]
03.
04.     encontrado = False
05.     Enquanto não encontrado faça:
06.         Se (menor = maior) então:
07.             Se (elementoDoMeio ≠ valorProcurado) então:
08.                 Retorne Nulo
09.             Senão
10.                 encontrado = True
11.         Senão Se (elementoDoMeio = valorProcurado) então:
12.             encontrado = True
13.         Senão Se (elementoDoMeio > valorProcurado) então:
14.             novaPosicao = indiceDoElementoDoMeio - 1
15.             maior = novaPosicao
16.             indiceDoElementoDoMeio = (menor + maior) // 2
17.             elementoDoMeio = listaOrdenada[indiceDoElementoDoMeio]
18.         Se (elementoDoMeio = valorProcurado) então:
19.             encontrado = True
20.         Senão Se (elementoDoMeio < valorProcurado) então:
21.             novaPosicao = indiceDoElementoDoMeio + 1
22.             menor = novaPosicao
23.             indiceDoElementoDoMeio = (menor + maior) // 2
24.             elementoDoMeio = listaOrdenada[indiceDoElementoDoMeio]
25.         Se (elementoDoMeio = valorProcurado) então:
26.             encontrado = True
27.     Retorne indiceDoElementoDoMeio
```



Conforme a análise empírica dos dois algoritmos, percebe-se que ambos apresentam complexidade logarítmica, ou seja, assintoticamente eles são praticamente iguais. No entanto, aparentemente, o algoritmo iterativo apresenta um desempenho um pouco melhor por utilizar loops simples, já o algoritmo recursivo, por fazer múltiplas chamadas de função, ele precisa alocar espaço para as variáveis locais e empilhar argumentos, causando um pequeno overhead comparado à execução de um loop simples.

Para as questões **12** e **13**, entregue os seguintes itens considerando o algoritmo implementado para resolver o problema computacional:

- Uma captura de tela que mostre a compilação correta na plataforma de teste;
- O cálculo da complexidade no tempo usando notação assintótica; e
- Um gráfico ilustrando a análise empírica, ou seja, a relação valor de entrada x tempo de execução.

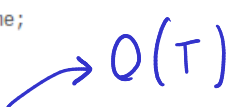
12. Resolva o seguinte problema computacional:

Problema: Ajude a Federação (#1588)

<https://www.beecrowd.com.br/judge/pt/problems/view/1588>

SUBMISSÃO # 35749672	
PROBLEMA:	1588 - Ajude a Federação
RESPOSTA:	Accepted
LINGUAGEM:	C++17 (g++ 7.3.0, -std=c++17 -O2 -lm) [+0s]
TEMPO:	0.029s
TAMANHO:	2,79 KB
MEMÓRIA:	-
SUBMISSÃO:	24/09/2023 18:48:48

CÓDIGO FONTE	
1	#include <iostream>
2	#include <string>
3	#include <map>
4	#include <cmath>
5	#include <vector>
6	
7	using namespace std;
8	
9	typedef struct Time{
10	string nomeDoTime;
11	int pontos;
12	int vitorias;
13	int gols;
14	int ordem;
15	}
16	Time;
17	
18	bool comparaTimes(const Time& time1, const Time& time2){
19	if (time1.pontos != time2.pontos) return time1.pontos > time2.pontos;
20	if (time1.vitorias != time2.vitorias) return time1.vitorias > time2.vitorias;
21	if (time1.gols != time2.gols) return time1.gols > time2.gols;
22	return time1.ordem < time2.ordem;
23	}
24	
25	int main() {
26	
27	int T, N, M, gols1, gols2;
28	string time1, time2, nomeDoTime;
29	
30	
31	cin >> T;
32	for (int i = 0; i < T; i++){
33	map<string, Time> listaDeClassificacao;
34	cin >> N >> M;
35	

 $O(T)$

```

36 ~ for (int i = 0; i < N; i++){ → O(N)
37 ~ // cout << "Nome do time: \n";
38 ~ cin >> nomeDoTime;
39 ~ Time time;
40 ~ time.nomeDoTime = nomeDoTime;
41 ~ time.gols = 0;
42 ~ time.pontos = 0;
43 ~ time.vitorias = 0;
44 ~ time.ordem = i;
45 ~ listaDeClassificacao[nomeDoTime] = time;
46 ~ }
47 ~
48 ~ for (int i = 0; i < M; i++){ → O(M)
49 ~ cin >> gols1 >> time1 >> gols2 >> time2;
50 ~ if (gols1 - gols2 > 0){
51 ~     listaDeClassificacao[time1].gols += gols1;
52 ~     listaDeClassificacao[time1].pontos += 3;
53 ~     listaDeClassificacao[time1].vitorias += 1;
54 ~     listaDeClassificacao[time2].gols += gols2;
55 ~ }else if (gols1 - gols2 < 0){
56 ~     listaDeClassificacao[time2].gols += gols2;
57 ~     listaDeClassificacao[time2].pontos += 3;
58 ~     listaDeClassificacao[time2].vitorias += 1;
59 ~     listaDeClassificacao[time1].gols += gols1;
60 ~ }else{
61 ~     listaDeClassificacao[time2].gols += gols2;
62 ~     listaDeClassificacao[time2].pontos += 1;
63 ~     listaDeClassificacao[time1].gols += gols1;
64 ~     listaDeClassificacao[time1].pontos += 1;
65 ~ }
66 ~ }
67 ~
68 ~ vector<Time> timesOrdenados;
69 ~ for (auto& par : listaDeClassificacao) {
70 ~     timesOrdenados.push_back(par.second);
71 ~ }
72 ~
73 ~ int tamanho = timesOrdenados.size();
74 ~ for (int i = 0; i < tamanho-1; i++) {
75 ~     for (int j = 0; j < tamanho-i-1; j++) {
76 ~         if (!(comparaTimes(timesOrdenados[j], timesOrdenados[j+1]))) {
77 ~             Time aux = timesOrdenados[j];
78 ~             timesOrdenados[j] = timesOrdenados[j+1];
79 ~             timesOrdenados[j+1] = aux;
80 ~             // swap(arr[j], arr[j+1]);
81 ~         }
82 ~     }
83 ~ }
84 ~ for (int i = 0; i < tamanho; i++){
85 ~     cout << timesOrdenados[i].nomeDoTime << "\n";
86 ~ }
87 ~
88 ~
89 ~ return 0;
90 ~ }

```

listaDeClassificacao tem o tamanho da quantidade de times N
 também tem tamanho N
 Bubble Sort
 } O(N)

O(N²)

Como o código está todo dentro do Loop for que itera até T, então a complexidade vai ser dada por:

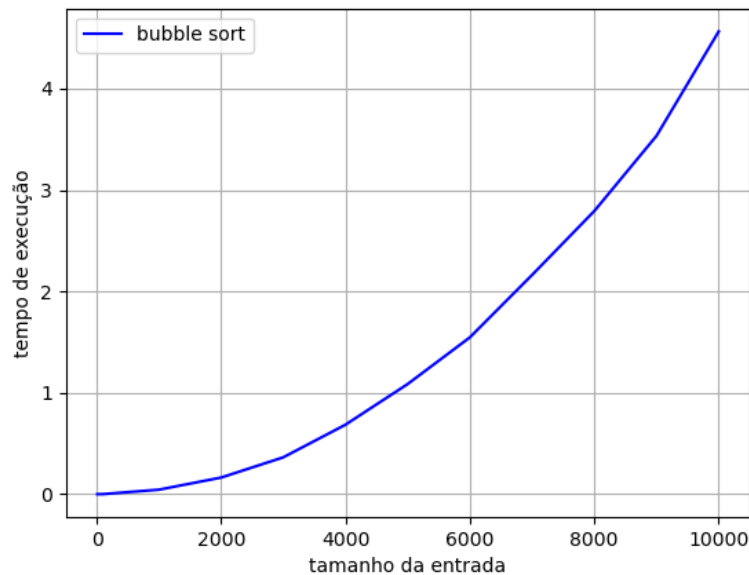
$$O(T) \cdot O(N + M + N + N^2 + N)$$

$$O(T) \cdot O(N^2 + \cancel{N} + M)$$

$$O(T) \cdot O(N^2 + N + M)$$

$$O(T \cdot (N^2 + N + M))$$

Considerando que a parte dominante do algoritmo é a ordenação dos times, podemos simplificar a complexidade para: $O(N^2)$



13. Resolva o seguinte problema computacional de forma **recursiva**:

Problema: A Lenda de Flavious Josephus (#1030)

<https://www.beecrowd.com.br/judge/pt/problems/view/1030>

SUBMISSÃO # 35757891	
PROBLEMA:	1030 - A Lenda de Flavious Josephus
RESPOSTA:	Accepted
LINGUAGEM:	C++17 (g++ 7.3.0, -std=c++17 -O2 -lm) [+0s]
TEMPO:	0.221s
TAMANHO:	1,63 KB
MEMÓRIA:	-
SUBMISSÃO:	25/09/2023 09:03:21

CÓDIGO FONTE	
<pre> 1 #include <iostream> 2 #include <cstdlib> 3 4 using namespace std; 5 6 typedef struct pessoa *Circulo; 7 8 struct pessoa{ 9 struct pessoa *prox; 10 int numeracao; 11 int salto; 12 }; 13 14 Circulo cria_pessoa(int numeracao, int salto){ 15 Circulo novaPessoa; 16 novaPessoa = (struct pessoa *)malloc(sizeof(struct pessoa)); 17 novaPessoa->prox = novaPessoa; // Inicializa o próximo como ele mesmo 18 novaPessoa->numeracao = numeracao; 19 novaPessoa->salto = salto; 20 21 return novaPessoa; 22 } 23 24 void add_pessoa(Circulo ultimo, int numeracao, int salto){ 25 Circulo novaPessoa = cria_pessoa(numeracao, salto); 26 Circulo proxUltimo = ultimo->prox; 27 28 novaPessoa->prox = proxUltimo; 29 ultimo->prox = novaPessoa; 30 } 31 32 void remover_pessoa(Circulo circ){ 33 Circulo pessoaRemovida = circ->prox; 34 circ->prox = pessoaRemovida->prox; 35 free(pessoaRemovida); 36 } 37 38 int main (){ </pre>	<div style="position: relative; height: 400px;"> <div style="position: absolute; right: 0; top: 10%; font-size: 4em;">}</div> <div style="position: absolute; right: 0; top: 10%; font-size: 2em;">O(1)</div> <div style="position: absolute; right: 0; top: 40%; font-size: 4em;">}</div> <div style="position: absolute; right: 0; top: 40%; font-size: 2em;">O(1)</div> <div style="position: absolute; right: 0; top: 70%; font-size: 4em;">}</div> <div style="position: absolute; right: 0; top: 70%; font-size: 2em;">O(1)</div> </div>

```

39 int NC;
40 int n;
41 int k;
42 int numeracaoFinal;
43 cin >> NC;
44
45 for (int i = 0; i < NC; i++){
46     cin >> n >> k;
47
48     Circulo circulo = cria_pessoa(1, k); // Começando a numeracao de 1
49
50     for (int i = 2; i <= n; i++){
51         add_pessoa(circulo, i, k);
52         circulo = circulo->prox;
53     }
54     while (circulo->prox != circulo) {
55         for (int i = 1; i < k; i++) {
56             circulo = circulo->prox;
57         }
58         remover_pessoa(circulo);
59     }
60
61     cout << "Case " << i+1 << ": " << circulo->numeracao << "\n";
62     free(circulo); // Liberando a última pessoa
63
64 }
65
66 return 0;
67 }

```

$\rightarrow O(NC + 1) \rightarrow O(NC)$
 $\rightarrow O(1)$
 $O(n)$
 $O(n.k)$ { leva n iterações para eliminar todos do círculo, até restar somente 1
 leva k iterações para percorrer o círculo de k em k
 $> O(1)$
 $\rightarrow O(1)$

Como o código está dentro do loop for que itera até NC, a complexidade do algoritmo vai ser dada por:

$$O(NC) \cdot O(\cancel{1} + \cancel{n} + n.k + \cancel{1} + \cancel{1})$$

$$O(NC) \cdot O(n.k)$$

$$O(NC \cdot n.k)$$

Como a parte dominante do código é a adição e remoção de pessoas no círculo e o loop for que itera até NC é apenas para aumentar os casos de teste por execução do código, podemos simplificar a complexidade para:

$$O(n + n.k)$$

$$O(n.k)$$

Análise assintótica da Lenda de Flavio Josephus

