# Alt F Trio
# UFPB

Gabriel Ayres, Gabriel Campelo e Gabriel Sherterton

21 de março de 2024

## Índice



## 1 Sorting

### 1.1 Bubble Sort

```
// compara  es: C(n) = O(n^2)
// trocas: T(n) = O(n^2)
// adptativo: O(n) quando vetor est  parcialmente ordenado

// Vantagens:
// algoritmo est vel
// simples

// Desvantagens:
// n o adapt vel
// muitas trocas

a33 void bubbleSort(int *ar, int n){
```

```
8f9     int i, aux;
011     bool troca;

016     do {
6fc         troca = false;
f50         for(int i = 0; i < n - 1; i++){
d7e             if(ar[i + 1] < ar[i]){
194                 aux = ar[i + 1];
89d                 ar[i + 1] = ar[i];
78a                 ar[i] = aux;
9c0                 troca = true;
9db             }
5df         }

51a     } while(troca);

505     return;
13c }
```

## 1.2 Inserction Sort

```
// compara  es C(n) = O(n^2)
// trocas T(n) = O(n^2)
// adaptativo O(n) quando vetor est  parcialmente ordenado

// Vantagens:
//    bom quando o vetor est  "quase" ordenado
//    um bom m todo quando se deseja adicionar poucos itens a um
    arquivo ordenado, pois o custo    linear

// Desvantagens:
// custo de compara  es O(n^2)
// custo de movimenta  es O(n^2)

492 void insertionSort(int *ar, int n){
141     int i, j , aux;
b4c     for(i = 1; i < n; i++){
28a         aux = ar[i];
37f         j = i - 1;
11c         while(j >= 0 && aux < ar[j]){
aa0             ar[j + 1] = ar[j];
237             j--;
10f         }
a9e         ar[j + 1] = aux;
64f     }
505     return;
71b }
```

## 1.3 Quick Sort

```
7fb void troca(int *ar, int i, int j){
133     int aux;
28a     aux = ar[i];
76b     ar[i] = ar[j];
b32     ar[j] = aux;
505     return;
d34 }

ded int particao(int *ar, int l, int r){
71d     int i = 0, fim = 0;
e7e     fim = l;
f5a     for(i = l + 1; i <= r; i++){
485         if(ar[i] < ar[l]){
83e             fim = fim + 1;
ebf             troca(ar, fim, i);
1dc         }
c41     }
a6b     troca(ar, l, fim);
645     return fim;
504 }

35a void quickSort(int *ar, int l, int r){
181     int i = 0;

572     if(l >= r) return;

073     i = particao(ar, l, r);
8cf     quickSort(ar, l, i - 1);
167     quickSort(ar, i + 1, r);

505     return;
d70 }
```

## 1.4 Selection Sort

```
// compara  es C(n) = O(n^2)
// trocas T(n) = O(n)

// Vantagens:
// bom quando a opera  o de trocar for muito cara
```

```
// Desvantagens:
// n o adapt vel
// n o    est vel

475 void selectionSort(int *ar, int n){
ab3     int i, j;
15b     int menor, indexMenor;

3f3     for(i = 0; i < n; i++){
f18         menor = ar[i];
8f7         for(j = i + 1; j < n; j++){
c69             if(ar[j] < menor){
4eb                 indexMenor = j;
6b5                 menor = ar[j];
5ee             }
bfb         }
fa5         ar[indexMenor] = ar[i];
b2e         ar[i] = menor;
d8e     }
505     return;
1a5 }
```

# 2 Matematica

## 2.1 Algoritmo de Euclides

```
// Time Complexity: O(log min(a,b))
// Auxiliary Space: O(log min(a,b))

eea int recursive_gcd(int a, int b){
650     if(b == 0) return a;
8c3     else return recursive_gcd(b, a % b);
bd9 }

ba4 int non_recursive_gcd(int a, int b){
1b4     while(b){
cae         a %= b;
257         swap(a, b);
22e     }
3f5     return a;
7c3 }
```

## 2.2 Algoritmo de Euclides Otimizado

```
// is an optimization to the normal Euclidean algorithm
// The slow part of the normal algorithm are the modulo operations.
// Modulo operations O(1)    , but are a lot slower than simpler
    operations like addition, subtraction or bitwise operations.

bce int binary_gcd(int a, int b) {
206     if(!a || !b) return a | b;
308     unsigned shift = __builtin_ctz(a | b);
2db     a >>= __builtin_ctz(a);
016     do {
cfd         b >>= __builtin_ctz(b);
f78         if (a > b)
257             swap(a, b);
064         b -= a;
788     } while(b);
c09     return a << shift;
232 }
```

## 2.3 Crivo de Erastotenes

```
// Time Complexity: O(nloglogn)
// Auxiliary Space: O(n)
// Find primes in range [2, n]

705 vector<int> sieve(int n){
e6e     vector<int> is_prime(n + 1, true);
19e     is_prime[0] = is_prime[1] = false;

bc4     for(int i = 2; (long long)i * i <= n; i++){
4a3         if(is_prime[i]){
985             for(int j = i * i; j <= n; j += i){
db3                 is_prime[j] = false;
f80             }
b3f         }
26e     }
054     return is_prime;
0c7 }
```

## 2.4 Euclides Extendido

```
// Teorema de B zout
```

```
// Time Complexity: O(log N)
// Auxiliary Space: O(log N)

// ax + by = gcd(a, b)
// gcd(a, b) = gcd(b % a, a) = (b % a) * x1 + a * y1
// ax + by = (b - (b/a) * a) * x1 + a * y1
// ax + by = a(y1 - (b/a) * x1) + b * x1
// x = y1 - (b/a) * x1
// y = x1

e4b int gcdExtended(int a, int b, int *x, int *y) {
220     if(a == 0){
b9d         *x = 0;
288         *y = 1;
73f         return b;
420     }

608     int x1, y1; // To store results of recursive call
c2d     int gcd = gcdExtended(b%a, a, &x1, &y1);

        // Update x and y using results of
        // recursive call
98c     *x = y1 - (b/a) * x1;
9bf     *y = x1;

e06     return gcd;
059 }
```

## 2.5 Euler's totient function

```
// Time Complexity: O(sqrt(n))
3fc int phi(int n){
efa     int result = n;
2ed     for(int i = 2; i * i <= n; i++){
c06         if(n % i == 0){
b52             int count = 0;
4cd             while(n % i == 0){
135                 n /= i;
157             }
21c             result -= result / i;
850         }
741     }
726     if(n > 1) result -= result / n;
dc8     return result;
8e9 }
```

```
// Euler's totient function 1 to n in O(nlog(log(n)))
// use the same ideas as the Sieve of Eratosthenes
429 vector<int> phi_1_to_n(int n){
675     vector<int> vec(n + 1);
4e3     for(int i = 0; i <= n; i++)
716         vec[i] = i;
508     for(int i = 2; i <= n; i++){
10b         if(vec[i] == i){
c6c             for(int j = i; j <= n; j += i){
816                 vec[j] -= vec[j] / i;
502             }
196         }
352     }
9d8     return vec;
eea }
```

## 2.6 Exponenciacao Rapida

```
// result = a^b % m
// Time Complexity = O(log b)

d95 ll binpow(ll a, ll b, ll m){
df2     ll result = 1;
63a     while(b > 0){
00f         if(b & 1) result *= a % m;
26c         a *= a % m;
1b4         b >>= 1;
4ea     }
dc8     return result;
6e8 }
```

## 2.7 Inversa Modular

```
// The exact time complexity of the this recursion is not known.
// It's is somewhere between  O (logm / loglogm) and O(m^(1/3 -2/177
   + e))
// demo:
// m prime and a,r < m -> exist a_inv and r_inv
// m = k*a + r
// 0     k*a + r (mod m)
// -k*a     r (mod m)
// -k     r*a_inv (mod m)
// a_inv     k*r_inv (mod m)
```

```
a18 int inv(int a, int m){
033     return a <= 1 ? a : m - (long long)(m / a) * inv(m % a, m) % m;
5fc }

// Binary Exponentation method
// O(log m)
// if a and m are relatively prime and m is prime
// power(a, m - 2)     a_inv (mod m)
951 long long binpow(long long a, long long b){
3fe     long long result = 1;
63a     while(b > 0){
427         if(b & 1) result *= a;
70c         a *= a;
1b4         b >>= 1;
aa4     }
dc8     return result;
4ad }

// precompute the inverse for every number in the range [1, m- 1] in
    O(m)
e8d int main(){
7f4     int m = 1000000007;
641     int invArray[m];
7d1     invArray[1] = 1;
92c     for(int a = 2; a < m; a++){
b75         invArray[a] = m - (long long)(m / a) * invArray[m % a] % m;
463     }
c20 }
```

### 2.8 Minimo Multiplo Comum

```
// Time Complexity: O(log min(a,b))
// Auxiliary Space: O(log min(a,b))
// lcm(a, b) * gcd(a, b) = a * b

ebb int lcm(int a, int b){
        // return a * b / __gcd(a, b) could be overflow
c27     return a / __gcd(a, b) * b;
312 }
```

### 2.9 Método de Horner para Avaliação Polinomial

```
// f(x) = (Cn * x^n) + (Cn-1 * x^n-1) + (Cn-2 * x^n-2) + ... + (C1 *
    x) + C0
/* Ex:
ecd  f(x) = 2x3 - 6x2 + 2x - 1
649  poly = {2, -6, 2, -1}
4f0  x = 3 -> f(3) = 5
c4c */

//Time Complexity: O(n)
//Auxiliary Space: O(1)

628 int horner(vector<int> &poly, int x){
855     int result = poly[0];
bc6     int n = poly.size();

6f5     for(int i = 1; i < n; i++){
a32         result = result * x + poly[i];
27a     }

dc8     return result;
f2a }
```

## 3 Problemas

### 3.1 Intersecao de Retangulos

```
2b7 #include <bits/stdc++.h>

ca4 using namespace std;

ef2 struct Rect {
619     int x1, y1, x2, y2;
0ac     int area(){
065         return (y2 - y1) * (x2 - x1);
b37     }
985 };

c93 int intersect(Rect p, Rect q){
6a6     int xOverlap = max(0, min(p.x2, q.x2) - max(p.x1, q.x1));
931     int yOverlap = max(0, min(p.y2, q.y2) - max(p.y1, q.y1));
1e5     return xOverlap * yOverlap;
e02 }
```

### 3.2 Permutacoes de String

```
// CSES - Creating Strings
// https://cses.fi/problemset/task/1622/

2b7 #include <bits/stdc++.h>

ca4 using namespace std;

d25 string str;
f6a vector<string> perms;
e41 int char_count[26];

497 void search(const string &curr = ""){
532     if(curr.size() == str.size()){
5b5         perms.push_back(curr);
505         return;
d12     }
42f     for(int i = 0; i < 26; i++){
962         if(char_count[i] > 0){
19c             char_count[i]--;
efd             search(curr + (char)('a' + i));
411             char_count[i]++;
818         }
08b     }
9ac }

e8d int main(){
912     cin >> str;
e28     for(char c : str){
f9d         char_count[c - 'a']++;
b4a     }

ff4     search();

161     cout << perms.size() << endl;
63a     for(int i = 0; i < perms.size(); i++){
efc         cout << perms[i] << endl;
657     }

bb3     return 0;
e7e }
```

### 3.3 Problema das 8 Damas

```
2b7 #include <bits/stdc++.h>
f3d #define _ ios_base::sync_with_stdio(0);cin.tie(0);
42a #define endl '\n'
efe #define pb push_back
ca4 using namespace std;

1a8 int n;
ac9 int cnt = 0;

890 void add(vector<bool> &cols, vector<bool> &diag1, vector<bool> &diag2, int row, int col) {
3ba     cols[col] = true;
fd8     diag1[row - col + n - 1] = true;
5de     diag2[row + col] = true;
294 }

b21 void rem(vector<bool> &cols, vector<bool> &diag1, vector<bool> &diag2, int row, int col) {
bfe     cols[col] = false;
9fb     diag1[row - col + n - 1] = false;
8b3     diag2[row + col] = false;
d15 }

4a5 void backtracking(int row, vector<bool> &cols, vector<bool> &diag1, vector<bool> &diag2) {
e99     if(row == n) {
b8d         cnt += 1;
505         return;
a21     }
27b     for(int col = 0; col < n; col++) {
a5b         if(!cols[col] && !diag1[row - col + n - 1] && !diag2[row + col]) {
b88             add(cols, diag1, diag2, row, col);
8c7             backtracking(row + 1, cols, diag1, diag2);
2c6             rem(cols, diag1, diag2, row, col);
ef2         }
4d7     }
826 }

f77 int main(){ _
a68     cin >> n; // number of rows = columns
a4f     vector<bool> cols(n, false), diag1(2 * n - 1, false), diag2(2 * n - 1, false);
72c     backtracking(0, cols, diag1, diag2);
0a9     cout << cnt << endl;

bb3     return 0;
24f }
```

## 3.4 Quadrado Perfeito

```
2b7 #include <bits/stdc++.h>

// Time Complexity: O(log n)
// Auxiliary Space: O(1)
d5a bool isPerfectSquare(long double n){
884     if(n >= 0){
            // se a raiz de n    inteira n o haver   arredondamento
               para o menor inteiro
569         long long sr = sqrt(n);

            // verifica se houve o arredondamento e retorna a reposta
9b6         return (sr * sr == n);
9b3     }

        //retorna falso caso x < 0
d1f     return false;
e0b }

// Time Complexity: O(log n)
// Auxiliary Space: O(1)
c38 bool binary_isPerfectSquare(long long n){
        // caso 0 e 1
8e8     if(n <= 1) return true;

        // limites da busca bin ria
fb9     long long l = 1, r = n;
e47     long long square, mid;
3d5     while(l <= r){
            // calcular valor do meio
b7b         mid = (l + r) / 2;

            // calcular quadrado do termo do meio
f94         square = mid * mid;

e55         if(square == n){
8a6             return true;
cbc         }

            // buscar na direita
852         else if(square < n){
0dc             l = mid + 1;
fb0         }

            // buscar na esquerda
4e6         else {
982             r = mid - 1;
2e2         }
665     }

        // caso saia do loop sem achar um quadrado perfeito
d1f     return false;
f49 }


// Time Complexity: O(sqrt(n))
// Auxiliary Space: O(1)
d5a bool isPerfectSquare(long double n){
5c7     if(floor(sqrt(n)) == ceil(sqrt(n))) return true;
00b     else return false;
e67 }
```

## 3.5 Ruina do Jogador

```
2b7 #include <bits/stdc++.h>

ca4 using namespace std;

// p    a probabilidade de ganhar um turno
// q = 1 - p, ou seka, a probabilidade de perder um turno
// Ri    a quantia inicial de dinheiro
// N    quantidade de dinheiro para ser vitorioso

773 double solve(double p, double q, int Ri, int N){
425     if(Ri == 0) return 0;
b21     if(Ri == N) return 1;

        // jogo justo
3b2     if(p == q) return (double)Ri / N;

        // p != q
4c3     return (1 - (double)pow(q / p, Ri)) / (1 - (double)pow(q / p,
    N));
afc }
```

# 4 Extra

## 4.1 template.cpp

```
#include <bits/stdc++.h>
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define pb push_back
#define all(x) (x).begin(), (x).end()

using namespace std;

typedef long long ll;
typedef unsigned long long llu;

int main(){ _
    int tt;
    cin >> tt;
    while(tt--) {

    }

    return 0;
}
```