Departamento de Computação

Programação Competitiva unesp

Programação Competitiva

Professores responsáveis: Rene Pegoraro e Wilson M Yonezawa

Coach das equipes: Pedro Henrique Paiola

Monitores: Arissa, Nicolas e Luis Henrique

Competições de programação

- Competições de Programação são provas com duração pré-determinada onde os participantes são desafiados a resolver uma série de problemas, usando linguagens de programação específicas.

- Os problemas são de entrada e saída textuais, em que o formato de ambas é bem definido.

- Os códigos são julgados automaticamente por um *software* juiz. Basicamente o juiz irá comparar a saída gerada por seu programa com a saída esperada.

Competições de programação

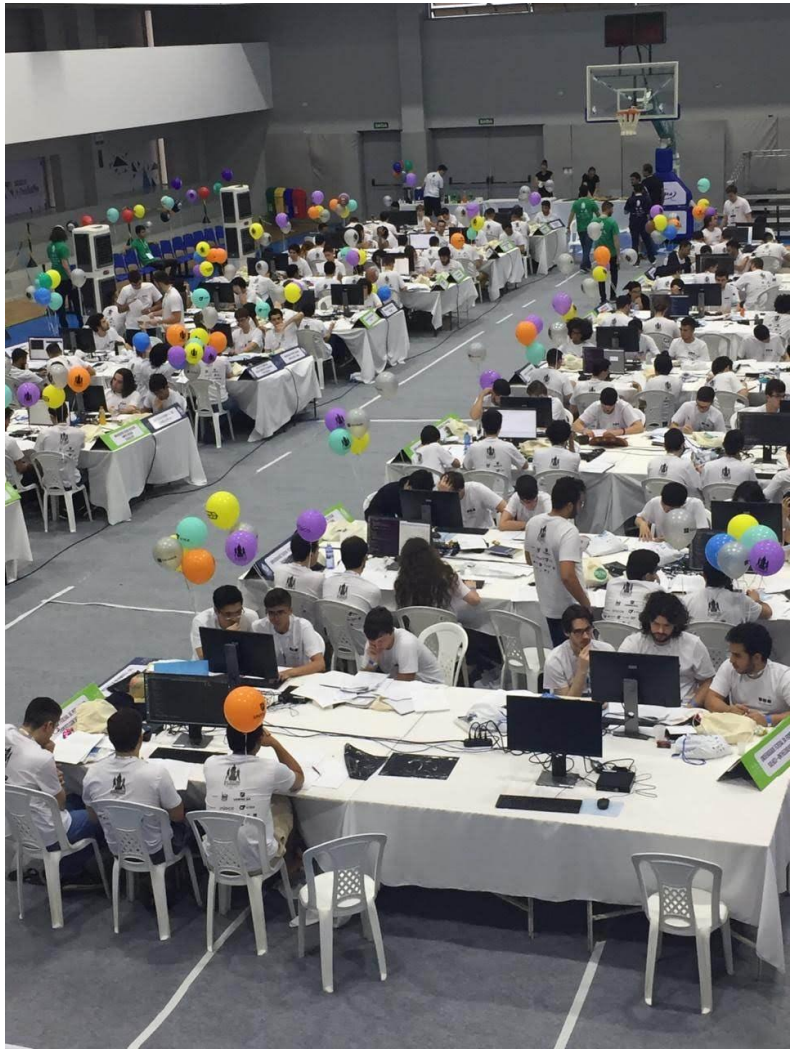
- Possíveis respostas do juiz:
  - AC – Accepted
  - WA – Wrong Answer
  - PE – Presentation Error
  - TLE – Time Limit Exceeded
  - RE – Runtime Error
  - CE – Compilation Error

Maratona SBC de Programação

- Evento da Sociedade Brasileira de Computação (SBC), realizado desde 1996.
- Destinada a alunos e alunos de cursos de graduação e início de pós na área de Computação e afins.
- Classificatória para as finais mundiais do concurso de programação, o International Collegiate Programming Contest.

Maratona SBC de Programação

- Ocorre em duas fases:
  - Fase Regional: normalmente em setembro
  - Final Brasileira: normalmente em novembro
- Times de 3 pessoas
- 1 computador por grupo
- Permitido consulta de material impresso
- 5 horas de *contest*
- Em torno de 13 problemas a serem resolvidos



Por quê?

1. Conhecimento e resolução de problemas
   - Você é confrontado com diversos problemas, todos muito diferentes, envolvendo diversas técnicas e estruturas de dados específicas.
   - A maioria dos problemas vão além do óbvio, requisitando o domínio de diversos conhecimentos e criatividade para combiná-los e aplicá-los adequadamente.
   - Estudando programação competitiva você irá aprender assuntos, de forma prática e aplicada, que o curso de graduação só irá oferecer mais tarde (ou nem irá oferecer).

Por quê?

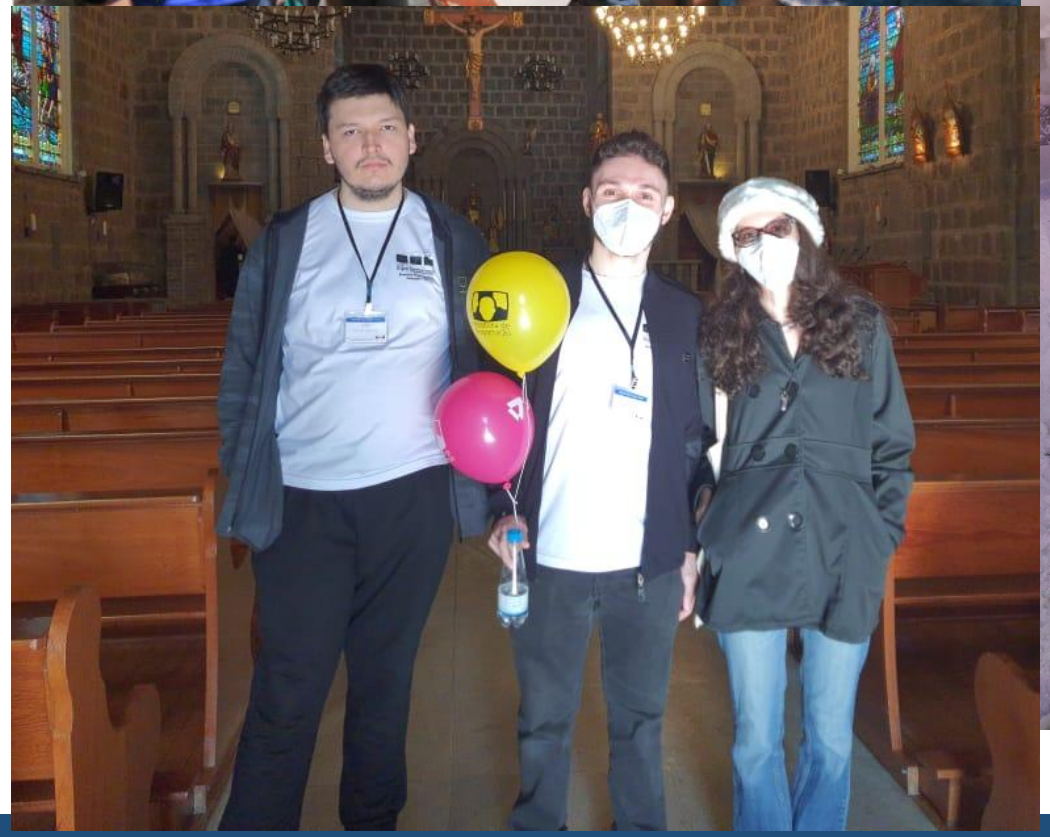
2. Trabalho em equipe
   - A Maratona de Programação é obrigatoriamente feita em grupos de três pessoas.
   - O grupo terá que aprender a lidar com a escassez de recursos e tempo para resolver o maior número de problemas durante a competição.
   - Uma boa dinâmica de grupo é tão essencial quanto o conhecimento técnico dos competidores individuais.

Por quê?

3. Mercado de Trabalho
   - Grandes empresas valorizam a participação em competições de programação.
   - As últimas Fases Nacionais da Maratona, por exemplo, foram patrocinadas por empresas como Google, Microsoft e B2W.
   - A Google organiza sua própria competição de programação, a Google Code Jam, voltada para identificar talentos para um potencial emprego na mesma.
   - Preparo para entrevistas técnicas de programação das grandes empresas

Por quê?

4. Viagens
   - Passeios
   - Comida
5. *Networking*
6. Brindes
7. Balões



Por quê?


Patrocinadores:
arcasolutions
finch
soluções
partnerfusion
Ferragut

"Observe que ser bem versado em programação competitiva não é objetivo final, apenas o meio. A verdadeira meta é produzir programadores/cientistas da computação versáteis, que estejam muito mais preparados para produzir *softwares* melhores ou para enfrentar complicados problemas de pesquisa de Ciência da Computação no futuro"

(Steven & Felix Halim, tradução nossa)

Outras competições

- OBI
- Google Code Jam
- Code Jam to I/O for Women
- Meta Coding Competitions
- Reply Challenge

- Plataformas de contest:
  - Beecrowd
  - Codeforces
  - AtCoder
  - TopCoder
  - E muitos outros :)

Bibliografia

- Felix Halim, Steven Halim. **Competitive Programming 3**.
- Steven Skiena. **Programming Challenges:** The Programming Contest Training Manual.
- Antti Laaksonen. **Guide to Competitive Programming:** Learning and Improving Algorithms Through Contests
- Steven Skiena. **The Algorithm Design Manual.**

Outros recursos

- CP-Algorithms: uma enciclopédia de diversos algoritmos em C++, com tópicos bem explicados.
- Geeks for Geeks: diversos artigos sobre programação e computação como um todo.
- Neps Academy: contém cursos, problemas e plataformas para discussão. Diversos conteúdos possuem acesso gratuito, outros necessitam a realização de uma assinatura.
- CSES: traz conjuntos de problemas para diversos temas, tratando uma variedade de problemas clássicos e recorrentes.

Outros recursos

- Canais do YouTube:
  - Programação Competitiva UNESP: nosso canal no YouTube, com as gravações das aulas de 2020 e 2021.
  - GEMA ICMC: canal do Grupo de Estudos para a Maratona de Programação (GEMA) do ICMC – USP São Carlos.
  - MaratonUSP: canal do grupo de estudos do IME-USP.

Nosso treinamento

- Na UNESP Bauru, estamos reorganizando nosso treinamento desde 2018. Com a pretensão de potencializar e agilizar o aprendizado de nossos alunos.
  - Um objetivo secundário, que já alcançamos parcialmente, é que os próprios alunos sejam responsáveis pelo treinamento.

- Como é feito hoje
  - 2º ano: disciplinas de Laboratório de Programação Competitiva I e II
  - 3º e 4º ano: treinamento, aperfeiçoamento e monitoria
  - Oficinas
  - Simulados

Nosso treinamento

- Próximos objetivos:
  - Treinamento para o 1º ano.
    - Destaque particular para a OBI.
  - Maior destaque para outras competições, além da Maratona.
  - Aproximação com outras instituições de ensino da região.
  - Projeto de extensão
    - Treinamento destinado para alunos do Ensino Médio: ETEC e CTI
    - Formalização do nosso trabalho

Nosso treinamento

- O que estudamos?
  - Paradigmas de Projeto de Algoritmos
    - Força Bruta
    - Backtracking
    - Algoritmos gulosos
    - Divisão e conquista
    - Programação Dinâmica
  - Estruturas de dados
    - Pilha, fila, fila de prioridades
    - Árvores
    - Grafos
  - Outros...
    - Teoria dos Números
    - Geometria computacional
    - Processamento de Strings

Nosso treinamento: LPC I

1. Introdução e aquecimento
2. Introdução ao C++ STL – Sort, vector, queue e stack
3. Busca binária – C++ STL (lower_bound, upper_bound, map, set, priority_queue)
4. Força Bruta – Backtracking
5. Algoritmo Guloso – Divisão e Conquista
6. Programação Dinâmica I
7. Programação Dinâmica II

Nosso treinamento: LPC I

8. Árvores
9. Disjoint-set (union-find)
10. Introdução à Teoria dos Grafos
11. Grafos: Problema do Caminho Mínimo
12. Grafos: Árvore Geradora Mínima (MST)
13. Strings: KMP e String Hashing
14. Teoria dos Números
15. Análise combinatória

Nosso treinamento: LPC II

1. Busca ternária
2. Grafos: Pontes e Ordenação Topológica
3. Grafos: Fluxo Máximo
4. Grafos: Problema da Coloração e Emparelhamento Máximo
5. Strings: Suffix Trie, Tree & Array
6. Geometria computacional
7. Digit-DP

Nosso treinamento: LPC II

8. Programação Dinâmica com Bitmask e Bitset
9. PD: Convex Hull Trick
10. Teoria dos Jogos
11. Segment Tree
12. Árvore de Fenwick (BIT)
13. Sparse Table
14. HLD Decomposition
15. LCA

Escolhendo uma linguagem

- Na maratona, o sistema de submissão executado usa a distribuição Linux Ubuntu 20.04, com as seguintes linguagens de programação, incluindo seus compiladores/interpretadores, configuradas:
  - **C:** gcc versão 9.3.0
  - **C++17:** gcc versão 9.3.0
  - **Python:** Python 3.8.5
  - **Java:** openjdk-11
  - **Kotlin:** Kotlin versão 1.4

Escolhendo uma linguagem

- Entre as linguagens suportadas, as mais utilizadas são C++, Python e Java.

- Com qual linguagem devo começar?

- Nossa recomendação se baseia em uma **condicional simples**:
  - **Se** você já conhece e trabalhou com uma ou mais das linguagens citadas, utilize aquela com a qual esteja mais confortável;
  - **Senão**, comece aprendendo a linguagem mais popularzinha e adorada, C++.

Por que utilizar C++?

1. Usualmente, C++ é uma linguagem mais rápida que Java que, por sua vez, é mais rápida que Python, considerando a compilação e tempo de execução dos códigos.
2. Permite ao programador ter mais controle sobre sua implementação e sobre as partes da linguagem, como objetos, classes e templates, através de tipos parametrizados, visando implementações mais genéricas.
3. Proporciona o uso completo das bibliotecas C padrão, bem como das bibliotecas avançadas de C++, destacando entre elas a Standard Template Library (STL).

Resolvendo problemas com a linguagem

- Non-Integer Donuts foi o problema N da fase regional da XXV Maratona de Programação de 2019/2020.

Resolvendo problemas com a linguagem

Neil is a very important lawyer with a very important bank account. Since Neil is such a successful lawyer with many clients, he deposits money to his account every single morning.

After going to the bank and depositing money, Neil goes to work. And there lies Neil's great weakness: a donut shop. You see, Neil is a recovering donut addict, and although he hasn't eaten a donut in years, he can't help but wonder how many \$1.00 donuts he could buy with the money in his account if he were to relapse.

Having \$5.00 in his account means 5 donuts Neil could have, but what about \$4.50? Well, that is more than 4 donuts for sure, but definitely less than 5. How would one even buy a non-integer amount of donuts? That concept confuses Neil, so every time his account balance is not an integer, he stops to ponder the nature of non-integer donuts and ends up being late to work.

Now Neil has been late too many times and is starting to worry he will lose his job. He wants to know how many times he will be late to work during the next N days, given his initial account balance and the amount of money he will deposit each day. Please answer this for him, or else Neil will start pondering again.

Input

The first line contains an integer N ($1 \leq N \leq 1000$), the number of days Neil is interested in. Each of the next $N+1$ lines contains a string representing an amount of money. The first string is Neil's account initial balance, while the following N strings are the amounts Neil will deposit to his account in the different days. Each string has the form $\$X.Y$ where X is a substring of length 1 or 2 indicating the whole money in the amount $\$X.Y$, while Y is a substring of length exactly 2 denoting the cents in the amount $\$X.Y$. Both X and Y are made of digits, at least one of them contains a non-zero digit, and X does not have leading zeros.

Output

Output a single line with an integer indicating how many times Neil will be late to work during the following N days.

Resolvendo problemas com a linguagem

- Non-Integer Donuts foi o problema N da fase regional da XXV Maratona de Programação de 2019/2020.

- Nele, temos uma **quantia inicial X de centavos**. No **dia i**, **aumentamos** essa quantia em $\mathbf{a_i}$ **centavos** e, **se pudermos**, **transformamos os centavos em reais**. Isso acontece por **N dias**. O exercício pede como saída a **contagem dos dias** em que a **quantia de centavos é diferente de zero**.

- Link: N - Non-Integer Donuts

Non-Integer Donuts

Entrada:
4
$1.00
$0.01
$0.99
$10.00
$98.76

Saída:
2

Non-Integer Donuts - C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    char aux;
    int n, inteiro, decimal, centavos, atrasado = 0;
    cin >> n >> aux >> inteiro >> aux >> decimal;
    centavos = decimal;
```

Non-Integer Donuts - C++

```
    for (int i = 0; i < n; i++) {
        cin >> aux >> inteiro >> aux >> decimal;
        centavos = (centavos + decimal) % 100;

        if (centavos != 0) {
            atrasado++;
        }
    }

    cout << atrasado << "\n";

    return 0;
}
```

Non-Integer Donuts - Python

```python
n = int(input())
str = input()
centavos = int(str[str.find(".") + 1 : len(str)])
atrasado = 0

for i in range(n):
    str = input()
    decimal = int(str[str.find(".") + 1 : len(str)])
    centavos = (centavos + decimal) % 100

    if centavos != 0:
        atrasado = atrasado + 1

print(atrasado)
```

Non-Integer Donuts - Java

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();
        String str = sc.next();
        int ponto = str.indexOf("."), centavos = Integer.parseInt(str.substring(ponto + 1)),
atrasado = 0;
```

Non-Integer Donuts - Java

```
        for (int i = 0; i < n; i++) {
            str = sc.next();
            ponto = str.indexOf(".");
            int decimal = Integer.parseInt(str.substring(ponto + 1));
            centavos = (centavos + decimal) % 100;

            if (centavos != 0)
                atrasado++;
        }

        System.out.println(atrasado);
        sc.close();
    }
}
```

Como compilar e rodar os códigos?

- **C++**

  Compilar: g++ donuts.cpp -o donuts -g -O2 -std=gnu++17 -static (Opcional)

  Rodar: donuts < in > out

- **Python**

  Rodar: python3 donuts.py < in > out (Interpretador)

- **Java**

  Compilar: javac Main.java

  Rodar: java Main < in > out

C++: código base

```cpp
#include<bits/stdc++.h>

using namespace std;

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);                  //Se precisar de fast-io
}
```

C++: entrada padrão

```
int numA, numB;
double numC;
string name;

cin >> numA >> numB >> numC >> name;
```

C++: saída padrão

```
int a;
double c;

...

cout << “A: ” << a << endl;
cout << “C: ” << setprecision(3) << fixed << c << endl;
```

Python: entrada padrão

```
entrada = input().split()

numA = int(entrada[0])
numB = int(entrada[1])
numC = float(entrada[2])
name = entrada[3]
```

Python: saída padrão

```
entrada = input().split()
a = int(entrada[0])
c = float(entrada[1])


...


print(“A:”, a)
print(“C:”, "%.3f" % c)
```

Java: código base

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        //permite receber dados da entrada padrão
        Scanner sc = new Scanner(System.in);
    }
}
```

Java: entrada padrão

```
Scanner sc = new Scanner(System.in);
int numA, numB;
double numC;
String name;

numA = sc.nextInt();
numB = sc.nextInt();
numC = sc.nextDouble();
name = sc.next();
```

Java: saída padrão

```
int a;
double c;


...

System.out.println("A: " + a);
System.out.println("C: " + String.format("%.3f", c));
```

Situações comuns

- Ler até acabar o arquivo de entrada

```
//C++
int a, b, c;
while (cin >> a >> b) {
    c = a + b;
    cout << c << "\n";
}
```

Exemplo de entrada	Exemplo de saída
1 2 5 10 3 2	3 15 5

Situações comuns

- Ler até acabar o arquivo de entrada

```python
#Python
import sys
for linha in sys.stdin:
    entrada = linha.split()
    a = int(entrada[0])
    b = int(entrada[1])
    c = a + b
    print(c)
```

Exemplo de entrada	Exemplo de saída
1 2 5 10 3 2	3 15 5

Situações comuns

- Ler até acabar o arquivo de entrada

```
//Java
Scanner sc = new Scanner(System.in);
while (sc.hasNextInt()) {
    int a, b, c;
    a = sc.nextInt();
    b = sc.nextInt();
    c = a + b;
    System.out.println(c);
}
```

Exemplo de entrada	Exemplo de saída
1 2 5 10 3 2	3 15 5

Situações comuns

- Número de casos de teste predeterminado (Opção 1)

```
//C++
int t, a, b, c;
cin >> t;
for (int i = 0; i < t; i++) {
    cin >> a >> b;
    c = a + b;
    cout << c << "\n";
}
```

Exemplo de entrada	Exemplo de saída
3 1 2 5 10 3 2	3 15 5

Situações comuns

- Número de casos de teste predeterminado (Opção 1)

```python
#Python
t = int(input())
for i in range(t):
    entrada = input().split()
    a = int(entrada[0])
    b = int(entrada[1])
    c = a + b
    print(c)
```

Exemplo de entrada	Exemplo de saída
3 1 2 5 10 3 2	3 15 5

Situações comuns

- Número de casos de teste predeterminado (Opção 1)

```java
//Java
Scanner sc = new Scanner(System.in);
int t, a, b, c;
t = sc.nextInt();
for (int i = 0; i < t; i++) {
    a = sc.nextInt();
    b = sc.nextInt();
    c = a + b;
    System.out.println(c);
}
```

Exemplo de entrada	Exemplo de saída
3 1 2 5 10 3 2	3 15 5

Situações comuns

- Número de casos de teste predeterminado (Opção 2)

```
//C++
int t, a, b, c;
cin >> t;
while (t--) {
    cin >> a >> b;
    c = a + b;
    cout << c << "\n";
}
```

Exemplo de entrada	Exemplo de saída
3 1 2 5 10 3 2	3 15 5

Situações comuns

- Número de casos de teste predeterminado (Opção 2)

```python
#Python
t = int(input())
while t > 0:
    entrada = input().split()
    a = int(entrada[0])
    b = int(entrada[1])
    c = a + b
    print(c)
    t -= 1
```

Exemplo de entrada	Exemplo de saída
3 1 2 5 10 3 2	3 15 5

Situações comuns

- Número de casos de teste predeterminado (Opção 2)

```java
//Java
Scanner sc = new Scanner(System.in);
int t, a, b, c;
t = sc.nextInt();
while (t-- > 0) {
    a = sc.nextInt();
    b = sc.nextInt();
    c = a + b;
    System.out.println(c);
}
```

Exemplo de entrada	Exemplo de saída
3 1 2 5 10 3 2	3 15 5

Situações comuns

- Última entrada é marcada por zeros

```
//C++
int a, b, c;
while (1) {
    cin >> a >> b;
    if (a == 0 && b == 0) {
        break;
    }
    c = a + b;
    cout << c << "\n";
}
```

Exemplo de entrada	Exemplo de saída
1 2 5 10 3 2 0 0	3 15 5

Situações comuns

- Última entrada é marcada por zeros

```python
#Python
while 1:
    entrada = input().split()
    a = int(entrada[0])
    b = int(entrada[1])
    if a == 0 and b == 0:
        break
    c = a + b
    print(c)
```

Exemplo de entrada	Exemplo de saída
1 2 5 10 3 2 0 0	3 15 5

Situações comuns

- Última entrada é marcada por zeros

```java
//Java
Scanner sc = new Scanner(System.in);
int a, b, c;
while (true) {
    a = sc.nextInt();
    b = sc.nextInt();
    if (a == 0 && b == 0) {
        break;
    }
    c = a + b;
    System.out.println(c);
}
```

Exemplo de entrada	Exemplo de saída
1 2 5 10 3 2 0 0	3 15 5

Complexidade

- Algoritmo: conjunto finito de passos que para um dado conjunto de dados chega a solução de um problema.
- Um problema geralmente pode ser resolvido por diferentes algoritmos
- Como comparar esses diferentes algoritmos?

ANÁLISE DA COMPLEXIDADE DE ALGORITMOS

Complexidade

- Podemos expressar a eficiência de um algoritmo descrevendo o seu tempo de execução em função do tamanho da entrada (quantidade de dados).
- O que nos interessa é o comportamento assintótico do algoritmo, ou seja, quando o tamanho da entrada é muito grande ($n \to \infty$).

Complexidade

- Na prática, isso nos permite nos preocupar apenas com a ordem de grandeza do tempo de execução, e não a função exata que descreva esse tempo

$$\mathrm{f(n)} = 2\mathrm{n}^2 + 5n = O(n^2)$$
$$\mathrm{g(n)} = 500\mathrm{n} + 4000 = \mathrm{O(n)}$$

- Normalmente, vamos nos focar no pior caso do problema. Por exemplo: se estivermos analisando um algoritmo de busca em um vetor, não vamos analisar imaginando que o elemento buscado é o primeiro a ser verificado, mas sim o último, ou que ele nem esteja no vetor.

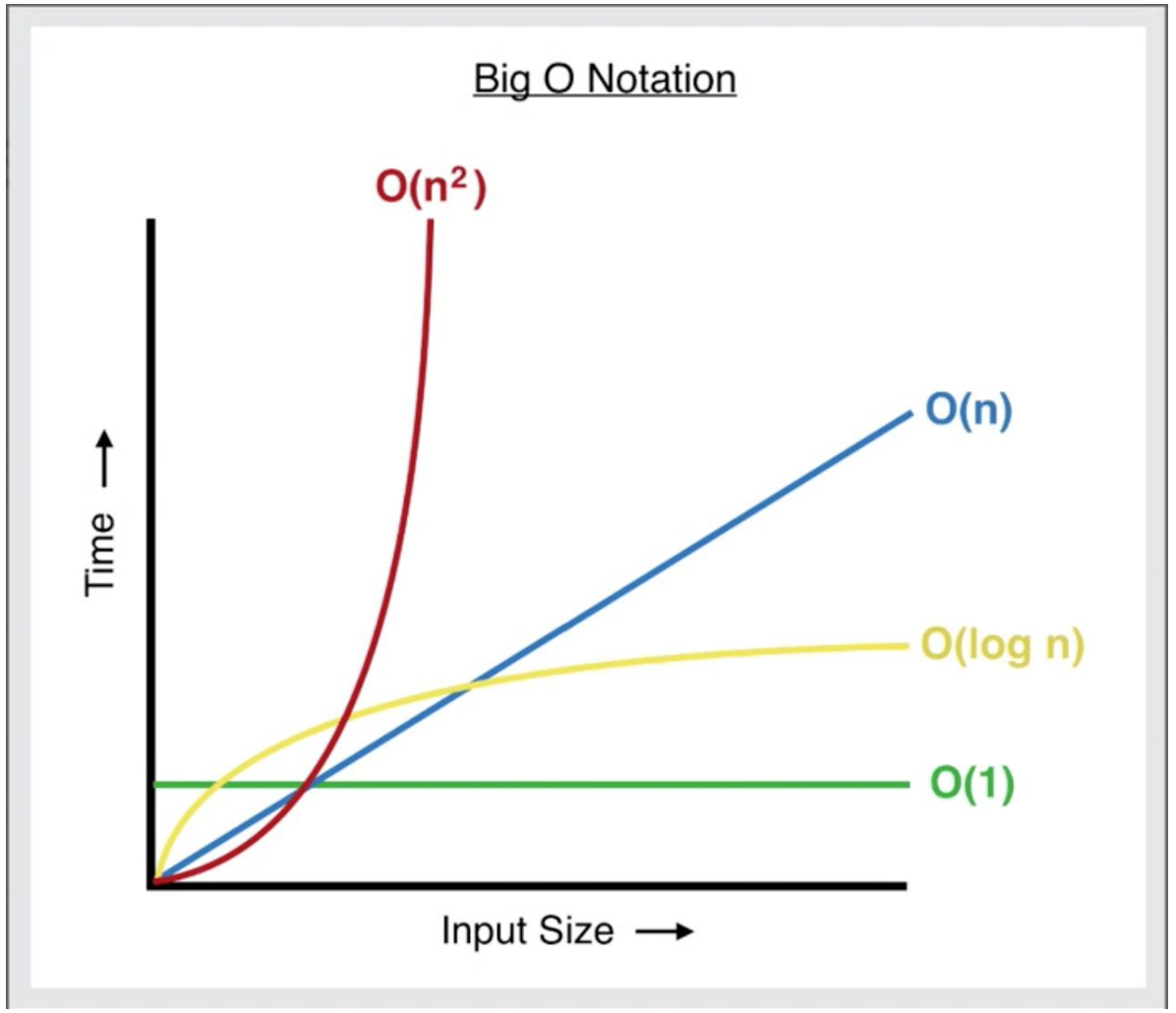
Complexidade

```
int buscaLinear(vector<int> vet, int x)
{
    for(int i = 0; i < vet.size(); i++)
    {
        if (vet[i] == x)
            return i;
    }
    return -1;
}
```

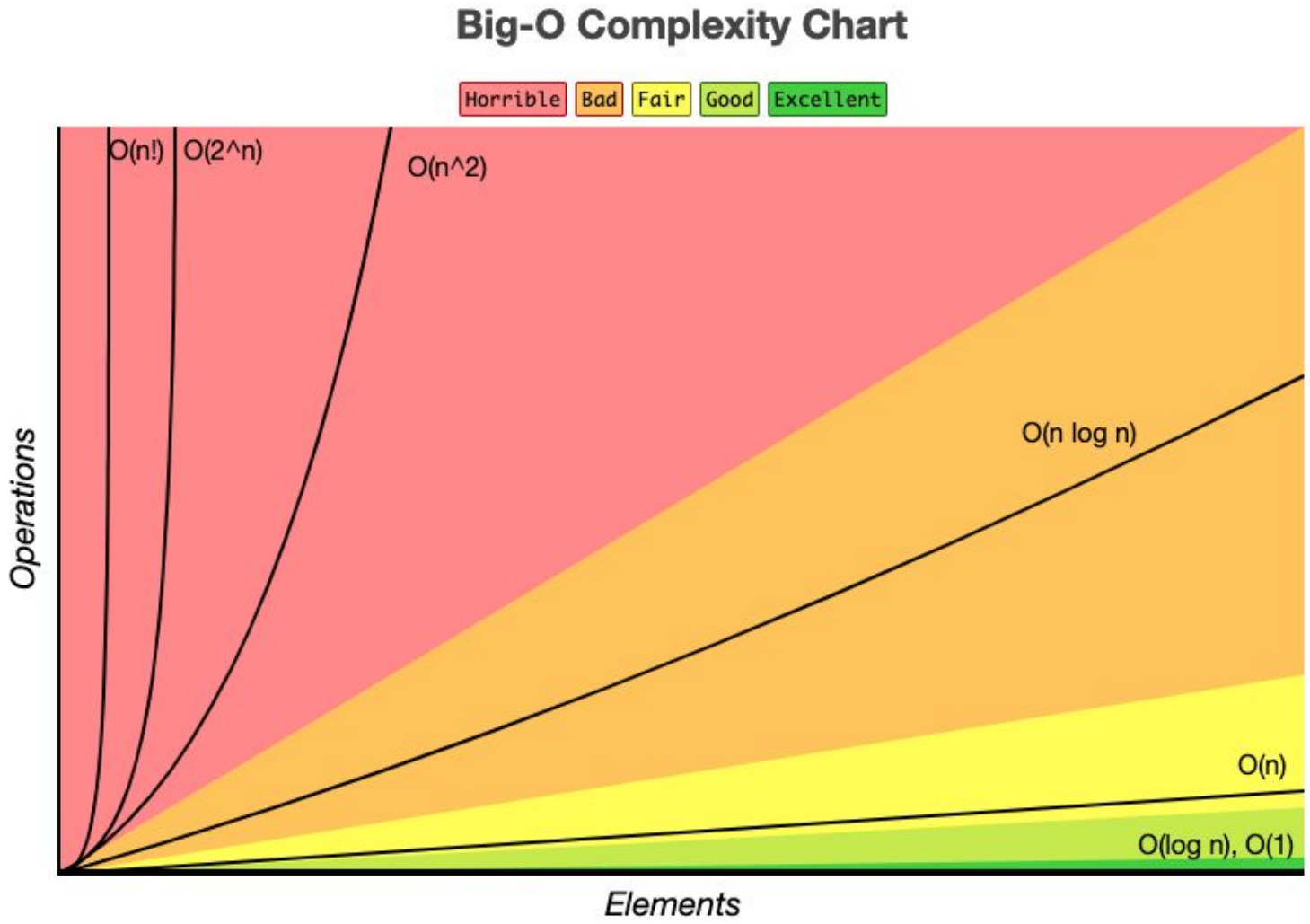
Complexidade

Complexidade	Terminologia
O(1)	Complexidade constante
O(log n)	Complexidade logarítmica
O(n)	Complexidade linear
O(n log n)	Complexidade n log n
$O(n^b)$	Complexidade polinomial
$O(b^n)$, b > 1	Complexidade exponencial
O(n!)	Complexidade fatorial

Complexidade



Complexidade



Complexidade

Tempo estimado para:	N=100	N=10000
O(log N)	10^{-8} segundos	10^{-7} segundos
O(N)	10^{-7} segundos	10^{-5} segundos
O(N log N)	10^{-6} segundos	10^{-3} segundos
$O(N^2)$	10^{-5} segundos	0.1 segundos
$O(N^3)$	10^{-3} segundos	16 minutos
$O(N^4)$	0.1 segundos	115 dias
$O(2^N)$	10^{13} anos	10^{2993} anos
O(N!)	10^{141} anos	10^{35642} anos

Complexidade

n	Worst AC Algorithm	Comment
$\leq [10..11]$	$O(n!), O(n^6)$	e.g. Enumerating permutations (Section 3.2)
$\leq [15..18]$	$O(2^n \times n^2)$	e.g. DP TSP (Section 3.5.2)
$\leq [18..22]$	$O(2^n \times n)$	e.g. DP with bitmask technique (Section 8.3.1)
≤ 100	$O(n^4)$	e.g. DP with 3 dimensions + $O(n)$ loop, ${}_nC_{k=4}$
≤ 400	$O(n^3)$	e.g. Floyd Warshall's (Section 4.5)
$\leq 2K$	$O(n^2 \log_2 n)$	e.g. 2-nested loops + a tree-related DS (Section 2.3)
$\leq 10K$	$O(n^2)$	e.g. Bubble/Selection/Insertion Sort (Section 2.2)
$\leq 1M$	$O(n \log_2 n)$	e.g. Merge Sort, building Segment Tree (Section 2.3)
$\leq 100M$	$O(n), O(\log_2 n), O(1)$	Most contest problem has $n \leq 1M$ (I/O bottleneck)

Table 1.4: Rule of thumb time complexities for the 'Worst AC Algorithm' for various single-test-case input sizes n, assuming that your CPU can compute $100M$ items in 3s.

Introdução ao STL

- O STL é uma biblioteca frequentemente incluída nos compiladores C++.
- Ela fornece elementos do tipo:
  - containers,
  - iteradores,
  - algoritmos e
  - functores.
- Eles podem ser utilizados com praticamente qualquer tipo de dado.
- Hoje veremos brevemente algumas das principais estruturas, e suas versões análogas em Java e Python.

Introdução ao STL

- A escolha de uma estrutura de dados adequada depende das características específicas desejadas:
  - Operações disponíveis
  - Complexidade de cada operação
- Tabela comparativa: www.cplusplus.com/reference/stl/

Introdução ao STL

Container	Stores	Over head	[ ]	Iterators	Insert	Erase	Find	Sort
list	T	8	n/a	Bidirect'l	C	C	N	N log N
deque	T	12	C	Random	C at begin or end; else N/2	C at begin or end; else N	N	N log N
vector	T	0	C	Random	C at end; else N	C at end; else N	N	N log N
set	T, Key	12	n/a	Bidirect'l	log N	log N	log N	C
multiset	T, Key	12	n/a	Bidirect'l	log N	d log (N+d)	log N	C
map	Pair, Key	16	log N	Bidirect'l	log N	log N	log N	C
multimap	Pair, Key	16	n/a	Bidirect'l	log N	d log (N+d)	log N	C
stack	T	n/a	n/a	n/a	C	C	n/a	n/a
queue	T	n/a	n/a	n/a	C	C	n/a	n/a
priority_ queue	T	n/a	n/a	n/a	log N	log N	n/a	n/a
slist	T	4	n/a	Forward	C	C	N	N log N

STL: String

- Em C, temos que ~~sofrer em~~ utilizar vetores de *char* (terminados por '\0') para representar *strings* (cadeias de caracteres).

- Declarando uma string:

```
char nome_string[size];
```

- Exemplos

```
char str[10];
char str[] = "String";
char str[] = {'S', 't', 'r', 'i', 'n', 'g', '\0'};
```

STL: String

- Em C++, a STL nos traz a classe string, que simplifica muito nosso trabalho, além de trazer uma série de métodos auxiliares.

- Declarando uma string:
  ```
  string nome_string;
  ```

- Exemplos
  ```
  string str;
  ```

STL: String

- Exemplo de benefício: strings podem ser comparadas normalmente utilizando os operadores <, >, <=, >=, == e !=

- Exemplo:

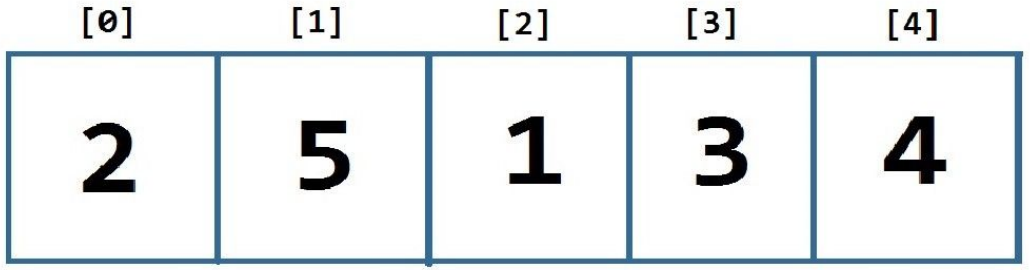
```
if (aluno[i] == “João”)
        cout << “João está na lista, na posição ” << i << endl;
```

STL: String

- Mais sobre strings:
  - Documentação C++
  - Artigo do GeeksforGeeks
- Strings em Python
  - W3schools: Python Strings
  - DevMedia: Tipos de dados em Python - String
- Strings em Java
  - DevMedia: String em Java
  - DevMedia: Trabalhando com string

STL: Vector

- Um vector é basicamente um vetor dinâmico que tem seu tamanho alterado automaticamente conforme elementos são inseridos ou removidos dele.



STL: Vector

- **Declarando um vector:**

```
vector<tipo_dado> nome_vetor;
```

- Exemplos

```
vector<int> vetA;
vector<double> vetB;
vector<pair<int, int> > vetC;
```

STL: Vector

- **Inserindo elemento no final do vector (push_back):**
  `nome_vetor.push_back(dado);`

- Exemplos

```
//vetA = {0, 4, 2}
vetA.push_back(5);
//vetA = {0, 4, 2, 5}

//vetC = {(0,2), (3, 4)}
vetC.push_back({8,2});
//vetC = {(0,2), (3, 4), (8,2)}
```

STL: Vector

- **Acesso ao i-ésimo elemento do vetor:**

```
nome_vetor[i];  //com i=0 sendo a primeira posição
```

Obs: se essa posição não existir no vector, teremos um erro em tempo de execução (Runtime error)

- Exemplos

```
//vetA = {0, 4, 2, 5}
cout << vetA[1] << endl; //Na tela será impresso: 4
vetA[2] = 3;
//vetA = {0, 4, 3, 5}
```

STL: Vector

- **Ordenando elementos do vetor:**

```
sort(nome_vetor.begin(), nome_vetor.end());
```

- Exemplo

```
//vetA = {0, 4, 5, 2}
sort(vetA.begin(), vetA.end());
//vetA = {0, 2, 4, 5}
```

STL: Vector

- **Funções úteis:**

  `nome_vetor.size()` - retorna a quantidade de elementos no vetor

  `nome_vetor.empty()` - retorna true se o vetor está vazio, false caso contrário

  `nome_vetor.clear()` - remove todos os elementos do vetor

STL: Vector

- **Complexidades:**

`nome_vetor.push_back()`	O(1)
`nome_vetor[i]` (Acesso aleatório)	O(1)
`nome_vetor.insert()`	O(n)
`nome_vetor.erase()`	O(n)
`nome_vetor.size()`	O(1)
`nome_vetor.empty()`	O(1)
`nome_vetor.clear()`	O(n)

STL: Vector

- Exemplo de problema: [Teams Forming (CodeForces 1092B)]

B. Teams Forming

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are n students in a university. The number of students is even. The i-th student has programming skill equal to a_i.

The coach wants to form $\frac{n}{2}$ teams. Each team should consist of exactly two students, and each student should belong to exactly one team. Two students can form a team only if their skills are equal (otherwise they cannot understand each other and cannot form a team).

Students can solve problems to increase their skill. One solved problem increases the skill by one.

The coach wants to know the minimum total number of problems students should solve to form exactly $\frac{n}{2}$ teams (i.e. each pair of students should form a team). Your task is to find this number.

Input

The first line of the input contains one integer n ($2 \le n \le 100$) — the number of students. It is guaranteed that n is even.

The second line of the input contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 100$), where a_i is the skill of the i-th student.

Output

Print one number — the minimum total number of problems students should solve to form exactly $\frac{n}{2}$ teams.

STL: Vector

- Exemplo de problema: Teams Forming (CodeForces 1092B)
  - Há *n* estudantes em uma universidade (*n* é sempre par), e o *i*-ésimo estudante possui uma habilidade de programação igual à a_i.
  - O técnico quer formar duplas entre todos os estudantes, mas cada membro da dupla precisa ter a mesma habilidade em programação.
  - Sempre que um estudante resolve um problema, sua habilidade sobe 1 nível.
  - Qual o número **mínimo** de problemas que os estudantes devem resolver para podermos formar todas as *n*/2 duplas.

STL: Vector

- Exemplo de problema: [Teams Forming (CodeForces 1092B)]

Examples

input	Copy
6 5 10 2 3 14 5	
output	Copy
5	

input	Copy
2 1 100	
output	Copy
99	

STL: Vector

- Exemplo de problema: Teams Forming (CodeForces 1092B)
  - Solução: gulosa, vamos unir os alunos que já possuem habilidades próximas, de forma que o estudante com menor habilidade tenha que realizar o menor número de exercícios para se igualar ao seu companheiro. Para isso, vamos ordenar o vetor e ver as diferenças de cada dupla.

5	10	2	3	14	5

STL: Vector

- Exemplo de problema: Teams Forming (CodeForces 1092B)
  - Solução: gulosa, vamos unir os alunos que já possuem habilidades próximas, de forma que o estudante com menor habilidade tenha que realizar o menor número de exercícios para se igualar ao seu companheiro. Para isso, vamos ordenar o vetor e ver as diferenças de cada dupla.

2	3	5	5	10	14

STL: Vector

- Exemplo de problema: Teams Forming (CodeForces 1092B)
  - Solução: gulosa, vamos unir os alunos que já possuem habilidades próximas, de forma que o estudante com menor habilidade tenha que realizar o menor número de exercícios para se igualar ao seu companheiro. Para isso, vamos ordenar o vetor e ver as diferenças de cada dupla.

resp = 1

STL: Vector

- Exemplo de problema: [Teams Forming (CodeForces 1092B)]
  - Solução: gulosa, vamos unir os alunos que já possuem habilidades próximas, de forma que o estudante com menor habilidade tenha que realizar o menor número de exercícios para se igualar ao seu companheiro. Para isso, vamos ordenar o vetor e ver as diferenças de cada dupla.

resp = 1 + 0

STL: Vector

- Exemplo de problema: Teams Forming (CodeForces 1092B)
  - Solução: gulosa, vamos unir os alunos que já possuem habilidades próximas, de forma que o estudante com menor habilidade tenha que realizar o menor número de exercícios para se igualar ao seu companheiro. Para isso, vamos ordenar o vetor e ver as diferenças de cada dupla.

resp = 1 + 0 + 4

STL: Vector

- Exemplo de problema: Teams Forming (CodeForces 1092B)
  - Solução: gulosa, vamos unir os alunos que já possuem habilidades próximas, de forma que o estudante com menor habilidade tenha que realizar o menor número de exercícios para se igualar ao seu companheiro. Para isso, vamos ordenar o vetor e ver as diferenças de cada dupla.

2	3	5	5	10	14

resp = 5

Solução em C++

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

using namespace std;

int main(){
    int n, i, aux;
    vector<int> a;
   cin >> n;
   for(i = 0; i < n; i++){
        cin >> aux;
        a.push_back(aux);
    }
    int resp = 0;
    for(i = 1; i < n; i += 2)
         resp += a[i] - a[i-1];
     cout << resp << "\n";
}
```

Solução em Python

```
n = int(input())
a = [int(x) for x in input().split()]
a = sorted(a)

resp = 0
for i in range(1, n, 2):
    resp = resp + a[i] - a[i-1]

print(resp)
```

Solução em Java

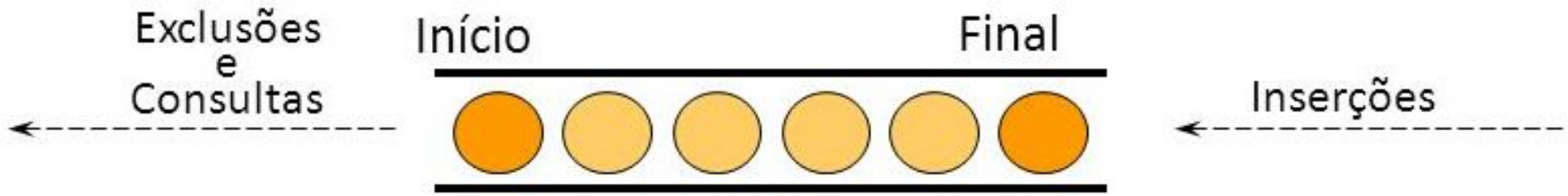
```
import java.util.Scanner;
import java.util.Collections;
import java.util.Vector;

public class teams {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int aux, i, n = in.nextInt();
        Vector<Integer> a = new Vector<Integer>(n);
```

Solução em Java

```
        for (i = 0; i < n; i++){
            aux = in.nextInt();
            a.add(aux);
        }
        Collections.sort(a);
        int resp = 0;
        for (i = 1; i < n; i += 2){
            resp += a.get(i) - a.get(i-1);
        }
        System.out.println(resp);
        in.close();
    }
}
```

STL: Queue

- A Queue é uma implementação de **fila**.
- Podemos pensar na **fila** como um vetor mas com as seguintes regras:
  - Podemos inserir elementos apenas no final da fila
  - Podemos acessar/retirar elementos apenas do início da fila
- **FIFO** (first in, first out)



STL: Queue

- **Declarando uma queue:**

```
queue<tipo_dado> nome_fila;
```

- Exemplos

```
queue<int> fila;
```

STL: Queue

- **Inserindo elemento no final da fila:**

```
nome_fila.push(x);
```

- Exemplos

```
//fila = {4, 2}
fila.push(5);
//fila = {4, 2, 5}
```

STL: Queue

- **Acessando elemento do início da fila:**
  `nome_fila.front();`
- **Removendo elemento do início da fila:**
  `nome_fila.pop();`

- Exemplos

```
//fila = {4, 2, 5}
cout << fila.front() << endl; //4
fila.pop();
//fila = {2, 5}
```

STL: Queue

- **Funções úteis:**

  `nome_fila.empty()` - retorna true se a fila está vazia, false caso contrário

  `nome_fila.size()` - retorna a quantidade de elementos na fila

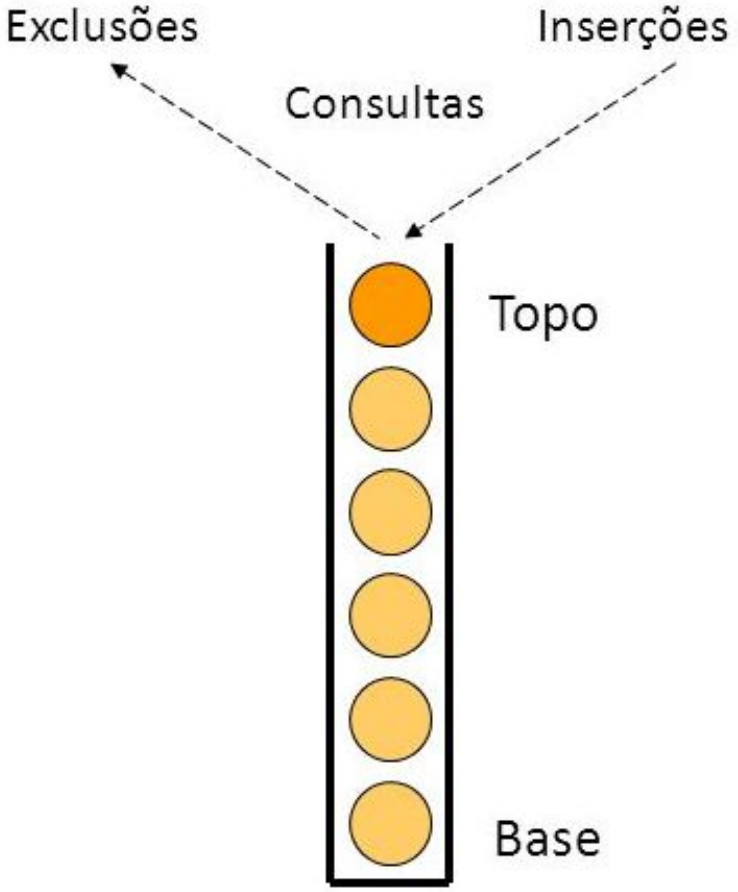
STL: Queue

- **Complexidades:**

`nome_fila.push()`	O(1)
`nome_fila.front()`	O(1)
`nome_fila.pop()`	O(1)
`nome_fila.empty()`	O(1)
`nome_fila.size()`	O(1)

STL: Stack

- Implementação de pilha.
- De forma semelhante à fila, podemos pensar na pilha como um vetor mas com as seguintes regras:
  - Podemos inserir elementos apenas no final da pilha
  - Podemos acessar/retirar elementos apenas do final da pilha
- LIFO (last in, first out)



STL: Stack

- **Declarando uma stack:**

```
stack<tipo_dado> nome_pilha;
```

- Exemplos

```
stack<int> pilha;
```

STL: Stack

- **Inserindo elemento no topo da pilha:**

```
nome_pilha.push(x);
```

- Exemplos

```
//pilha = {4, 2}
pilha.push(5);
//pilha = {4, 2, 5}
```

STL: Stack

- **Acessando elemento do topo da pilha:**

```
nome_pilha.top();
```

- **Removendo elemento do topo da pilha:**

```
nome_pilha.pop();
```

- Exemplos

```
//pilha = {4, 2, 5}
cout << pilha.top() << endl; //5
pilha.pop();
//pilha = {4, 2}
```

STL: Stack

- **Funções úteis:**

  `nome_pilha.empty()` - retorna true se a pilha está vazia, false caso contrário

  `nome_pilha.size()` - retorna a quantidade de elementos na pilha

STL: Stack

- **Complexidades:**

`nome_pilha.push()`	O(1)
`nome_pilha.top()`	O(1)
`nome_pilha.pop()`	O(1)
`nome_pilha.empty()`	O(1)
`nome_pilha.size()`	O(1)

STL: Stack

- Exemplo de problema: [Diamantes e Areia (BeeCrowd 1069)]

beecrowd | 1069

Diamantes e Areia

Por Neilor Tonin, URI Brasil

Timelimit: 1

João está trabalhando em uma mina, tentando retirar o máximo que consegue de diamantes "<>". Ele deve excluir todas as particulas de areia "." do processo e a cada retirada de diamante, novos diamantes poderão se formar. Se ele tem como uma entrada .<...<<..>>....>....>>>., três diamantes são formados. O primeiro é retirado de <..>, resultando .<...<>....>....>>>. Em seguida o segundo diamante é retirado, restando .<.......>....>>>. O terceiro diamante é então retirado, restando no final .....>>>., sem possibilidade de extração de novo diamante.

Entrada

Deve ser lido um valor inteiro **N** que representa a quantidade de casos de teste. Cada linha a seguir é um caso de teste que contém até 1000 caracteres, incluindo "<,>, ."

Saída

Você deve imprimir a quantidade de diamantes possíveis de serem extraídos em cada caso de entrada.

STL: Stack

- Exemplo de problema: Diamantes e Areia (BeeCrowd 1069)
  - Dada uma certa expressão, determinar o número de correspondências válidas de < e >
  - Exemplo:
    - .<...<<..>>....>....>>>
    - .<...<<..>>....>....>>>
    - .<...<<..>>....>....>>>
    - .<...<<..>>....>....>>>

STL: Stack

- Exemplo de problema: Diamantes e Areia (BeeCrowd 1069)

Exemplo de Entrada	Exemplo de Saída
2 <..><.<..>> <<<..<.....<<<<....>	3 1

STL: Stack

- Exemplo de problema: Diamantes e Areia (BeeCrowd 1069)
  - Solução: este é um problema clássico de pilha. Iremos percorrer a string dada pela entrada, empilhar cada ‘<’, e quando encontrarmos um ‘>’, se houver elementos na pilha, desempilhamos e incrementamos a resposta.

Solução em C++

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

using namespace std;

int main(){
    int n, i;
    string s;
    cin >> n;
    while(n--){
        cin >> s;
    stack<char> pilha;
    int resp = 0;
```

Solução em C++

```
        for(i = 0; i < s.size(); i++){
            if (s[i] == '<')
                pilha.push(s[i]);
            else if ((s[i] == '>') && !pilha.empty()){
                pilha.pop();
                resp++;
            }
        }
        cout << resp << "\n";
    }
}
```

Solução em Python

```
from collections import deque

n = int(input())
for k in range(n):
    s = input()
    pilha = deque()
    resp = 0
    for i in range(0, len(s)):
        if s[i] == '<':
            pilha.append(s[i])
        elif s[i] == '>' and len(pilha) > 0:
            pilha.pop()
            resp = resp + 1
    print(resp)
```

Solução em Java

```
import java.util.Scanner;
import java.util.Stack;

public class Main {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n, i;
        String s;
        n = in.nextInt();
        for(int k = 0; k < n; k++){
            s = in.next();
```

Solução em Java

```
            Stack<Character> pilha = new Stack<Character>();
            int resp = 0;
            for(i = 0; i < s.length(); i++){
                if (s.charAt(i) == '<')
                    pilha.push(s.charAt(i));
                else if (s.charAt(i) == '>' && !pilha.empty()){
                    pilha.pop();
                    resp++;
                }
            }
            System.out.println(resp);
        }
        in.close();
    }
}
```

STL: Map

- Um map armazena pares (chave, valor), sendo que as chaves e valores podem ser de qualquer tipo.
- A chave é utilizada para achar um elemento rapidamente.
- Diz-se, portanto, que um map “mapeia chaves para valores”
- Implementações comuns
  - Árvore (map)
  - Tabela Hash (unordered_map)

STL: Map

- **Declarando um map:**

```
map<tipo_chave, tipo_valor> nome_map;
unordered_map<tipo_chave, tipo_valor> nome_map2;
```

- Exemplo:

```
map<string, int> qtdeOcorrencias;
```

STL: Map

- **Inserindo elemento em um map:**
  `nome_map[chave] = valor;`

- Exemplo:
  `qtdeOcorrencias[“Olá”] = 4;`

STL: Map

- **Acessando elemento em um map:**
  `nome_map[chave]`

- Exemplo:
  ```
  cout << qtdeOcorrencias[“Olá”] << endl; //4
  ```

STL: Map

- **Verificando se uma chave está armazenada no map:**

  `nome_map.count(chave)`

  Retorna 1 se está, e 0 caso contrário.

- Exemplo:

```
if (qtdeOcorrencias.count("Olá"))
   cout << qtdeOcorrencias["Olá"] << endl;
else
   cout << "Palavra não encontrada" << endl;
```

STL: Map

- **Complexidades:**

	map	unordered_map
`map[chave] = valor`	O(log n)	O(1) ~ O(n)
`map[chave]`	O(log n)	O(1) ~ O(n)
`map.count(chave)`	O(log n)	O(1) ~ O(n)

STL: Map

- Exemplo de problema: [Ida à Feira (BeeCrowd 1281)]

beecrowd | 1281

Ida à Feira

Por Neilor Tonin, URI Brasil

Timelimit: 1

Dona Parcinova costuma ir regularmente à feira para comprar frutas e legumes. Ela pediu então à sua filha, Mangojata, que a ajudasse com as contas e que fizesse um programa que calculasse o valor que precisa levar para poder comprar tudo que está em sua lista de compras, considerando a quantidade de cada tipo de fruta ou legume e os preços destes itens.

Entrada

A primeira linha de entrada contém um inteiro **N** que indica a quantidade de idas à feira de dona Parcinova (que nada mais é do que o número de casos de teste que vem a seguir). Cada caso de teste inicia com um inteiro **M** que indica a quantidade de produtos que estão disponíveis para venda na feira. Seguem os **M** produtos com seus preços respectivos por unidade ou Kg. A próxima linha de entrada contém um inteiro **P** ($1 \leq P \leq M$) que indica a quantidade de diferentes produtos que dona Parcinova deseja comprar. Seguem **P** linhas contendo cada uma delas um texto (com até 50 caracteres) e um valor inteiro, que indicam respectivamente o nome de cada produto e a quantidade deste produto.

Saída

Para cada caso de teste, imprima o valor que será gasto por dona Parcinova no seguinte formato: R$ seguido de um espaço e seguido do valor, com 2 casas decimais, conforme o exemplo abaixo.

STL: Map

- Exemplo de problema: Ida à Feira (BeeCrowd 1281)
  - Dada o preço de uma lista de produtos (identificados por seus nomes = strings), e uma lista de compra com a quantidade comprada de cada produto, determinar o gasto total.
  - A solução é direta, porém o map ajuda implementá-la de forma fácil e eficiente

STL: Map

- Exemplo de problema: Ida à Feira (BeeCrowd 1281)

Exemplo de Entrada	Exemplo de Saída
2 4 mamao 2.19 cebola 3.10 tomate 2.80 uva 2.73 3 mamao 2 tomate 1 uva 3 5 morango 6.70 repolho 1.12 brocolis 1.71 tomate 2.80 cebola 2.81 4 brocolis 2 tomate 1 cebola 1 morango 1	R$ 15.37 R$ 15.73

Solução em C++

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

using namespace std;

int main(){
    int t, i;
    cin >> t;
    while(t--){
        int n, m, qtde;
        map<string, double> precos;
        string item;
        double valor, total = 0;
```

Solução em C++

```
        cin >> n;
        for(i = 0; i < n; i++){
            cin >> item >> valor;
            precos[item] = valor;
        }
        cin >> m;
        for(i = 0; i < m; i++){
            cin >> item >> qtde;
            total += precos[item] * qtde;
        }
        printf("R$ %.2lf\n", total);
    }
}
```

Solução em Python

```
t = int(input())
for k in range(t):
    precos = {}
    total = 0
    n = int(input())
    for i in range(n):
        item, valor = input().split()
        precos[item] = float(valor)
    m = int(input())
    for i in range(m):
        item, qtde = input().split()
        total = total + precos[item] * int(qtde)
    print("R$ {:.2f}".format(total))
```

Solução em Java

```
import java.util.Scanner;
import java.util.HashMap;
import java.util.Locale;
import java.util.Map;

public class Main {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in).useLocale(Locale.US);
        int t, i;
        t = in.nextInt();
        for(int k = 0; k < t; k++){
            int n, m, qtde;
            Map<String, Double> precos = new HashMap<String, Double>();
            String item;
```

Solução em Java


```
            Double valor, total = 0.;
            n = in.nextInt();
            for(i = 0; i < n; i++){
                item = in.next();
                valor = in.nextDouble();
                precos.put(item, valor);
            }
            m = in.nextInt();
            for(i = 0; i < m; i++){
                item = in.next();
                qtde = in.nextInt();
                total += precos.get(item) * qtde;
            }
            System.out.println("R$ " + String.format("%.2f", total));
        }
        in.close();
    }
}
```

STL: Set

- Um set é um container que armazena valores de uma maneira ordenada.
- As operações mais comuns em um set, são a inserção e a deleção de elementos:
- A implementação interna do set usa uma árvore de busca binária.
- Sets não permitem adicionar valores repetidos

STL: Set

Declarando um Set:

C++

```
set < tipo das variáveis > meu_set ;
```

Python

```
meu_set = set()
```

Java

```
SortedSet <tipo das variáveis> meu_set = new TreeSet<String> ();
```

STL: Set

Inserindo em um Set:

C++

```
meu_set.insert (elemento) ;
```

Python

```
meu_set.add (elemento);
```

Java

```
meu_set.add (elemento);
```

STL: Set

Deletando de um Set:

C++

```
meu_set.erase (elemento);
```

Python

```
meu_set.discard (elemento);
```

Java

```
meu_set.remove (elemento);
```

STL: Set

Verificando se um elemento está no Set:

C++

```
if ( meu_set.count(elemento) )
```

Python

```
if elemento in meu_set:
```

Java

```
if( meu_set.contains(elemento) )
```

STL: Set

Complexidades:

Inserir: O(log N)

Remover: O(log N)

Buscar: O(log N)

STL: Set

Algumas funções extras dos Sets:

Sets in C++

Sets in Java

Sets in Python

STL: Set

- **Exemplo de problema: Pérolas (BeeCrowd 1975)**

VI é uma professora de cálculo muito excêntrica, sempre que corrige as provas dos alunos (Que por sinal são provas dificílimas), ela anota todas as pérolas que encontra enquanto corrige, para que no dia da entrega ela possa escrever todas no quadro, para deixar os alunos envergonhados e que eles nunca mais errem as mesmas coisas.

Sempre que a bronca termina e as provas são entregues, os alunos tentam descobrir quem foi que teve mais pérolas no quadro. Como a cada prova os números de pérolas aumentam e os alunos tem que estudar muito pois a cada semana acontece uma nova prova de cálculo, eles não tem tempo para verificar todas as provas e ver quem apareceu mais vezes no quadro.

Sabendo que você é programador eles pediram sua ajuda para mostrar qual foi o aluno que teve mais pérolas escritas no quadro naquele dia.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém três inteiros **P**, **A** e **R** ($1 \leq P, A, R \leq 10^4$), indicando respectivamente, o número de pérolas, número de alunos e a quantidade de respostas dadas por cada aluno. Segue **P** linhas com as pérolas escritas no quadro que terão no máximo 1000 caracteres. Em seguida terão **A** alunos, para cada aluno a primeira linha será seu nome com no máximo 100 caracteres minúsculos de 'a' até 'z', seguindo as **R** linhas mostrando suas respostas. A entrada termina quando **P** = **A** = **R** = 0, e não deve ser processada.

OBS: Ignore as possibilidades de haver entradas de alunos com o mesmo nome ou pérolas repetidas.

Saída

Para cada saída, você deverá imprimir o nome do aluno que teve mais aparições no quadro, em caso de empate seu programa deverá mostrar todos os alunos com mais aparições separados por vírgulas em ordem alfabética.

STL: Set

- **Exemplo de problema:** Pérolas (BeeCrowd 1975)
  - Dada uma lista de pérolas dada pela professora, e as respostas dada por cada aluno, determinar qual(is) aluno(s) cometeram mais pérolas.
  - Assim como o caso anterior, a solução é direta, porém utilizar um set torna a implementação mais fácil.

STL: Set

- **Exemplo de problema:** [Pérolas (BeeCrowd 1975)]

Exemplo de Entrada	Exemplo de Saída
4 3 3 1 + 1 = 3 ((x - 2) + (x - 2)) / (x - 2) = 1 4 / 0 = 0 n! = n^2 gabriel 5 + 5 = 10 1 + 1 = 3 4 / 0 = ERRO alexandre ((x - 2) + (x - 2)) / (x - 2) != 1 1 + 1 = 2 2 + 3 = 5 anne 1 + 1 = 3 4 / 0 = ERRO 2 + 3 = 5 0 0 0	anne, gabriel

Solução em C++

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

using namespace std;

int main(){
    int p, a, r, i, j;
    while(1){
        cin >> p >> a >> r;
        if (a == 0)
            break;
        int qtdePerolas, maxPerolas = 0;
        string resp, nome;
```

Solução em C++

```
set<string> perolas;
set<string> nomes;

getline(cin, resp);
for(i = 0; i < p; i++){
    getline(cin, resp);
    perolas.insert(resp);
}

for(i = 0; i < a; i++){
    getline(cin, nome);
    qtdePerolas = 0;
```

Solução em C++

```
        for(j = 0; j < r; j++){
            getline(cin, resp);
            if (perolas.count(resp))
                qtdePerolas++;
        }
        if (qtdePerolas > maxPerolas) {
            nomes.clear();
            nomes.insert(nome);
            maxPerolas = qtdePerolas;
        } else if (qtdePerolas == maxPerolas) {
            nomes.insert(nome);
        }
    }
```

Solução em C++

```
        for(auto it : nomes){
            if (it == *nomes.begin())
                cout << it;
            else
                cout << ", " << it;
        }
        cout << endl;
    }
}
```

Solução em Python

```
while True:
    p, a, r = (int(x) for x in input().split())
    if a == 0:
        break
    maxPerolas = 0
    perolas = set()
    nomes = set()
    for i in range(p):
        resp = input()
        perolas.add(resp)
    for i in range(a):
        nome = input()
        qtdePerolas = 0
```

Solução em Python

```
for j in range(r):
    resp = input()
    if resp in perolas:
        qtdePerolas += 1
if qtdePerolas > maxPerolas:
    nomes = set()
    nomes.add(nome)
    maxPerolas = qtdePerolas
elif qtdePerolas == maxPerolas:
    nomes.add(nome)
```

Solução em Python

```
nomes = sorted(list(nomes))
for it in nomes:
    if it == nomes[0]:
        print(it, sep='', end='')
    else:
        print(", ", it, sep='', end='')
print()
```

Solução em Java

```
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int p, a, r, i, j;
        while(true)
        {
            p = in.nextInt();
            a = in.nextInt();
            r = in.nextInt();
            if (a == 0)
                break;
            int qtdePerolas, maxPerolas = 0;
            String resp, nome;
```

Solução em Java

```
Set<String> perolas = new HashSet<String>();
SortedSet<String> nomes = new TreeSet<String>();
in.nextLine();
for(i = 0; i < p; i++){
    resp = in.nextLine();
    perolas.add(resp);
}
for(i = 0; i < a; i++){
    nome = in.nextLine();
    qtdePerolas = 0;
    for(j = 0; j < r; j++){
        resp = in.nextLine();
        if (perolas.contains(resp))
            qtdePerolas++;
    }
```

Solução em Java


```
                if (qtdePerolas > maxPerolas){
                    nomes.clear();
                    nomes.add(nome);
                    maxPerolas = qtdePerolas;
                } else if (qtdePerolas == maxPerolas) {
                    nomes.add(nome);
                }
            }
            Iterator<String> it = nomes.iterator();
            while(it.hasNext()){
                nome = it.next();
                if (it.hasNext())
                    System.out.print(nome+", ");
                else
                    System.out.println(nome);
            }
        }
        in.close();
}
```

STL: Priority Queue

- Seja **S** um conjunto de elementos, onde cada elemento tem uma prioridade **P** associada. Uma fila de prioridades é um tipo abstrato de dados que permite executar as seguintes operações sobre **S**:
  - Encontrar um elemento máximo de **S**
  - Extrair um elemento máximo de **S**
  - Inserir um novo elemento em **S**
- Nesse caso estamos falando de uma fila de prioridades de máximo, mas também podemos ter fila de prioridades de mínimo.

STL: Priority Queue

- A implementação normalmente é feita utilizando a estrutura **heap**.
- Além de podermos usar fila de prioridades para resolver diversos problemas da Maratona ou da OBI, essa estrutura também é usada em algoritmos célebres, e que também são usados nas competições, como o algoritmo de Dijkstra e o algoritmo de Prim.
- C++ STL: priority_queue

STL: Priority Queue

- **Declarando uma priority_queue:**

```
priority_queue<tipo_dado> nome_pq;
```

- Exemplos

```
priority_queue<int> heap;
```

STL: Priority Queue

- **Inserindo um elemento na priority_queue:**

```
nome_pq.push(x);
```

- Exemplos

```
//topo da heap = 8
heap.push(10);
//topo da heap = 10
heap.push(9);
//topo da heap = 10
```

STL: Priority Queue

- **Acessando topo da priority_queue:**

```
nome_pq.top();
```

- **Removendo topo da priority_queue:**

```
nome_pq.pop();
```

- Exemplos

```
//Considerando o exemplo do slide anterior
cout << heap.top() << endl; //10
fila.pop();
//todo da heap = 9
```

STL: Priority Queue

- **Métodos úteis:**

  `nome_pq.empty()` - retorna true se a heap está vazia, false caso contrário

  `nome_pq.size()` - retorna a quantidade de elementos na heap

STL: Priority Queue

- **Complexidades:**

`heap.push()`	O(log n)
`heap.top()`	O(1)
`heap.pop()`	O(1)
`heap.size()`	O(1)
`heap.empty()`	O(1)

STL: Priority Queue

- Exemplo de problema: Add All (UVA 10954)

10954 Add All

Yup!! The problem name reflects your task; just add a set of numbers. But you may feel yourselves condescended, to write a C/C++ program just to add a set of numbers. Such a problem will simply question your erudition. So, lets add some flavor of ingenuity to it.

Addition operation requires cost now, and the cost is the summation of those two to be added. So, to add 1 and 10, you need a cost of 11. If you want to add 1, 2 and 3. There are several ways

$1 + 2 = 3$, cost $= 3$ $3 + 3 = 6$, cost $= 6$ Total $= 9$	$1 + 3 = 4$, cost $= 4$ $2 + 4 = 6$, cost $= 6$ Total $= 10$	$2 + 3 = 5$, cost $= 5$ $1 + 5 = 6$, cost $= 6$ Total $= 11$

I hope you have understood already your mission, to add a set of integers so that the cost is minimal.

STL: Priority Queue

- Exemplo de problema: Add All (UVA 10954)
  - Dada uma lista de números, determinar o custo mínimo para somar todos ele, dois a dois.
  - O custo é definido como a soma acumulada do resultado de cada adição.
  - Exemplo: para os números 1, 2 e 3

$1 + 2 = 3$, cost $= 3$ $3 + 3 = 6$, cost $= 6$ Total $= 9$	$1 + 3 = 4$, cost $= 4$ $2 + 4 = 6$, cost $= 6$ Total $= 10$	$2 + 3 = 5$, cost $= 5$ $1 + 5 = 6$, cost $= 6$ Total $= 11$

STL: Priority Queue

- Exemplo de problema: Add All (UVA 10954)

Input

Each test case will start with a positive number, N ($2 \leq N \leq 5000$) followed by N positive integers (all are less than 100000). Input is terminated by a case where the value of N is zero. This case should not be processed.

Output

For each case print the minimum total cost of addition in a single line.

Sample Input	Sample Output
3 1 2 3 4 1 2 3 4 0	9 19

STL: Priority Queue

- Exemplo de problema: Add All (UVA 10954)
  - Para resolver este problema, vamos começar a realizar a soma pelos números menores.
  - Porém, não podemos esquecer que o resultado de cada soma também passa a integrar a lista de números, e deve ser adicionada a outro número no momento oportuno.
  - Logo, cada resultado deve ser adicionado a lista, e então continuamos o processo de somar os menores números, do menor pro maior, até restar apenas um número na lista.

STL: Priority Queue

- Exemplo de problema: Add All (UVA 10954)
  - Exemplo: [3, 4, 1, 2, 5]

[3, 4, 1, 2, 5]	
[1, 2, 3, 4, 5] → [3, 3, 4, 5]	custo = 3
[3, 3, 4, 5] → [4, 5, 6]	custo = 3 + 6 = 9
[4, 5, 6] → [6, 9]	custo = 9 + 9 = 18
[6, 9] → [15]	custo = 18 + 15 = 33
[15]	

STL: Priority Queue

- Exemplo de problema: Add All (UVA 10954)
  - Uma min priority queue irá nos ajudar a resolver isso:
    - Adicionamos todos os números na priority queue
    - Enquanto houver mais de um número
      - Retiramos os dois menores
      - Somamos os dois valores e adicionamos o resultado à priority e ao custo total

Solução em C++

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

using namespace std;

int main(){
    int n, i, v;
    while(1) {
        cin >> n;
        if (n == 0)
            break;
        priority_queue<int> pq;
        for(i = 0; i < n; i++){
            cin >> v;
            pq.push(-v);
      }
```

Solução em C++

```
        int resp = 0;
        while(pq.size() > 1) {
            int a = -pq.top();
            pq.pop();
            int b = -pq.top();
            pq.pop();
            resp += a+b;
            pq.push(-(a+b));
        }
        cout << resp << "\n";
    }
}
```

Solução em Python

```
import heapq

while True:
    n = int(input())
    if n == 0:
        break
    pq = [int(x) for x in input().split()]
    heapq.heapify(pq)
    resp = 0
    while len(pq) > 1:
        a = heapq.heappop(pq)
        b = heapq.heappop(pq)
        resp += a+b
        heapq.heappush(pq, a+b)
    print(resp)
```

Solução em Java

```
import java.util.Scanner;
import java.util.PriorityQueue;

public class Main {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n, i, v;

        while(true){
            n = in.nextInt();
            if (n == 0)
                break;
            PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
```

Solução em Java

```
            for(i = 0; i < n; i++){
                v = in.nextInt();
                pq.add(v);
            }
            int resp = 0;
            while(pq.size() > 1) {
                int a = pq.poll();
                int b = pq.poll();
                resp += a+b;
                pq.add(a+b);
            }
            System.out.println(resp);
        }
        in.close();
    }
}
```