Day 3

09 November 2022 20:26

C++ Basics ✓

#include <bits/stdc++.h> ⇒ STL containers

```
#include <bits/stdc++.h> // header File
using namespace std;

int main(){
    int a; cin>>a; // Input
    cout<<a;       // Print
}
```

#include <vector> ✗

Input → cin>> $\boxed{a}$

Print → cout

C++ STL

Standard Template Library

Algorithms

☆ 1. sort(startaddress, endaddress) ✓

2. binary-search(startaddress, endaddress, value to find) ✓

☆ 3. reverse(first-iterator, second-iterator) ✓

10/5

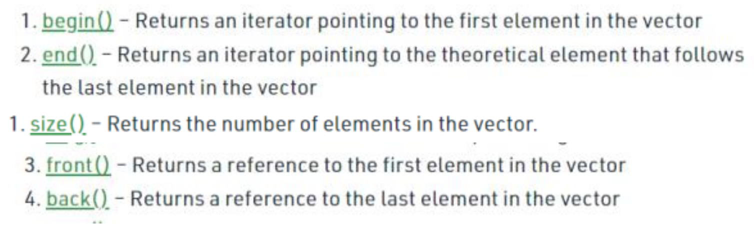
Containers

☆☆

[]

• vector

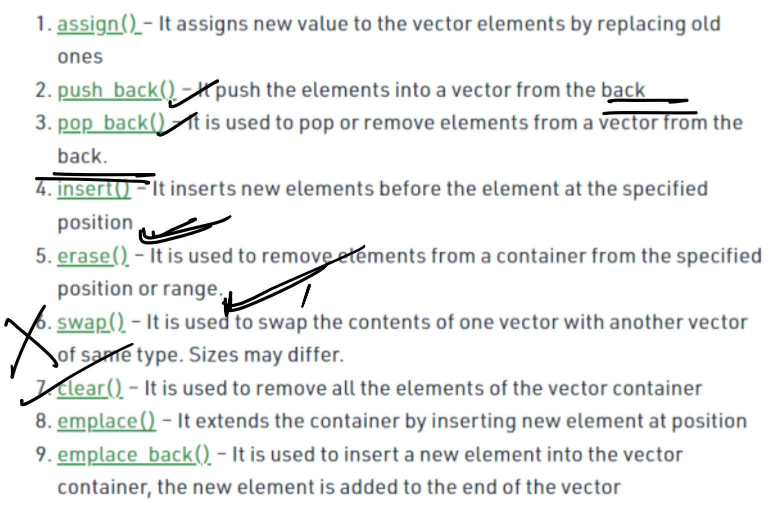
Dynamic size array



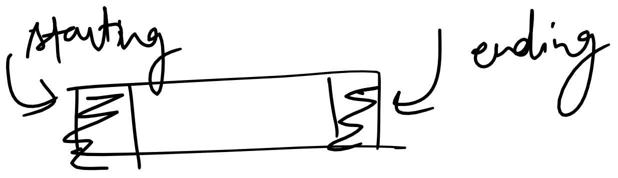
1. begin() – Returns an iterator pointing to the first element in the vector
2. end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector

1. size() – Returns the number of elements in the vector.

3. front() – Returns a reference to the first element in the vector
4. back() – Returns a reference to the last element in the vector



1. assign() – It assigns new value to the vector elements by replacing old ones
2. push_back() – It push the elements into a vector from the back ✓
3. pop_back() – It is used to pop or remove elements from a vector from the back. ✓
4. insert() – It inserts new elements before the element at the specified position ✓
5. erase() – It is used to remove elements from a container from the specified position or range. ✓
6. swap() – It is used to swap the contents of one vector with another vector of same type. Sizes may differ.
7. clear() – It is used to remove all the elements of the vector container
8. emplace() – It extends the container by inserting new element at position
9. emplace_back() – It is used to insert a new element into the vector container, the new element is added to the end of the vector



★★ Deque (Double ended queue) ←

★ insertion & deletion at both ends

push_front() ✓
push_back() ✓
pop_front() ✓
pop_back() ✓

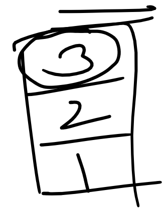
★★ queue (Implemented form of queue DS)

- push
- pop

- push ✓
- pop ✓
- front ✓

LIFO

☆ stack (Implemented form)
push ✓
pop ✓
top ✓



☆ priority_que (To use heap) ✓

☆ set → value sorted and unique

☆ multiset ✓ sorted by unique (multiple ocurence)

☆ map ✓ ⟶ { key, value }

☆ unordered_set

☆ unordered_map

☆ pair< Dt1 , Dt2 > → ✓

☆ vector (container) ⟶ Dynamic Array

#include <vector> ☆

↳ functionality

C++

int size;
cin >> size;
int arr[size];

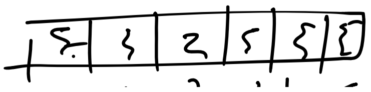
⊗ #include <bits/stdc++.h>

Initialise

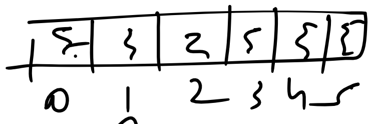
vector<int> arr(6, 0);

0	0	0	0	0	0

vector<int> arr; // Empty



vector<int> arr, [illegible]



Input

```
int size;              size = 6
cin >> size; vector<int> arr(size);
for(int i=0; i<size; i++){
        cin >> arr[i];
}
```

output

```
for(int i=0; i<size; i++){
    cout << arr[i];
}
```

* push_back()
* pop_back()
* size()

```
arr.push_back(6);

arr.pop_back();
```

$\boxed{\text{arr.size()}}$

set

| 1 | 1 | 1 | 2 | 2 | 3 | 3 |

⇓

S1 {1, 2, 3} ✓

unorderset
unique, ~~sorted~~
{2, 1, 3}

```
{ 1 : Avneesh
  2 : Utkarsh
}
```

map → keys sorted order

unordered_map:

{ 2: Uttkarsh
1: Avneesh }

key, value
pair (unique)
but not sorted ✓

[pair<int, string>]
{1 : "Priyanshu"}

Recursion

Recursion :- A function calls itself. X

```
void func1(){
    cout<< "Recursion" <<endl;
    func1();
}

int main(){
    func1();
}
```

Infinite
Recursion
Recursion
⋮
∞

★★★
Function का Activation Record को stack memory में save करता

stack overflow

ई || Stack

★★ Recursion :- A function calls itself to reduce some problem into smaller problem.

Print 1 to n

void printn(int n) $\longrightarrow$ 1 to n

Hypothesis

void printn(~~int~~ n) $\longrightarrow$ 1 to n

printn(n−1) $\longrightarrow$ 1 to n−1

print(n)

Recursion

① Hypothesis

② smaller version हमें क्या करके देगा

③ ~~ooo~~ बचा हुआ Part क्या करेंगे ।।

① printn(n) $\longrightarrow$ 1 to n

② printn(n−1) $\longrightarrow$ 1 to n−1 ④

③ print(n)

//Base Case

```
void printn(int n) {
    if(n<1) return;
    printn(n-1);  →  1 to n-1
```

printn(n-1); → 1 to n-1

cout<<n<<endl;

}

$\boxed{1 \quad 2 \quad 3 \quad 4 \quad 5}$

Order of Execution

n=5 printn(5) → 5

printn(4) ~~print(5)~~ → 4

print(3) → 3

print(2) → 2

print(1) → cout<<1<<endl;

~~print(0)~~

1 2 3 4 5

print n to 1

~~hypothesis~~

print2(n) → n to 1

print n

print2(n-1) → n-1 to 1

```
void print2(int n){ if(n<1) return;
    print(n);
    print2(n-1);
```

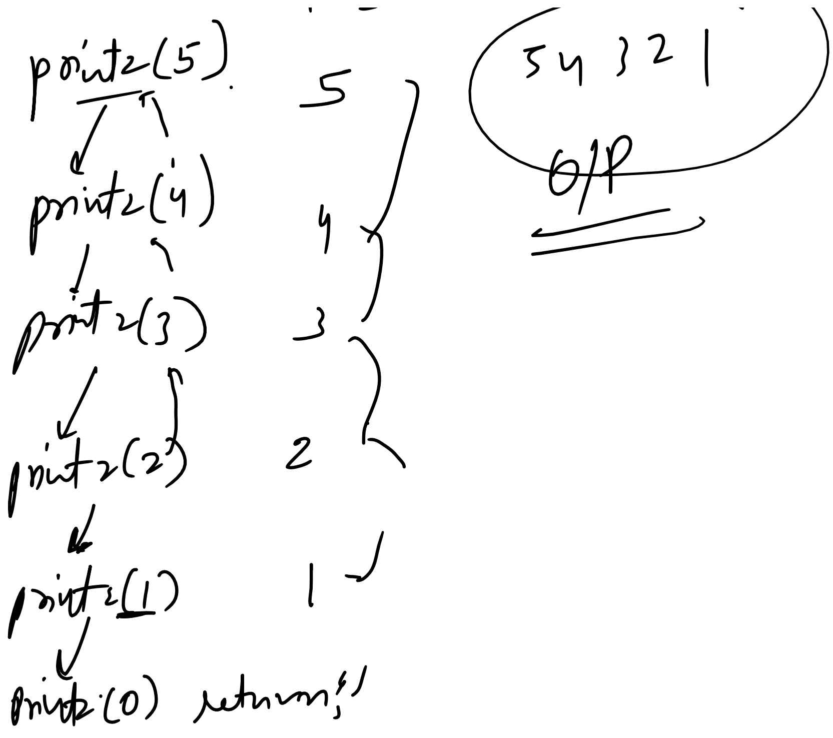
//Scope of }

function

n=5

print2(5) 5 ? 5 4 3 2 1

function



Easy

- Hypothesis
- smaller input
- Extra
- Base

Recursion → Choice + Decisions
Backtracking

Choice Diagram

Interview • Important

I/P "ABC" / [1, 2, 3]

O/P All subsets of this string

{1, 2, 3}

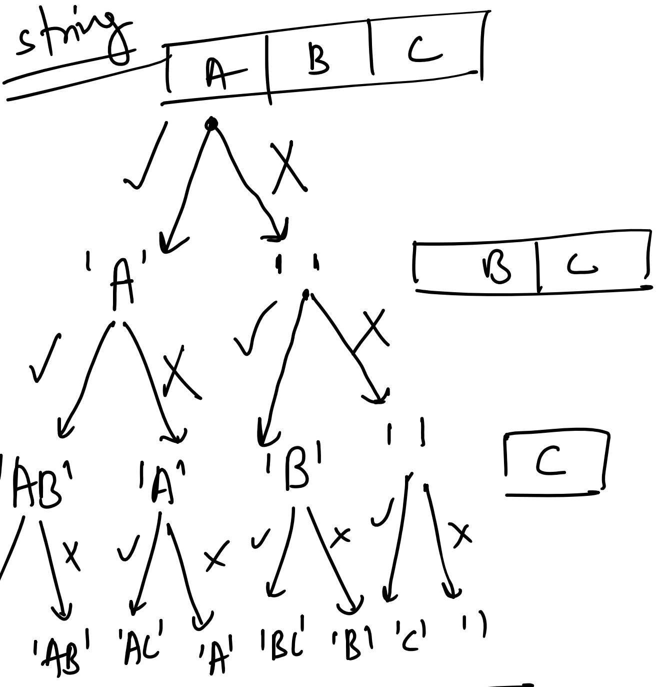
⇓ subsets

[{}, {1}, {2}, {3}, {1,2}, {2,3}, {3,1} {1,2,3}]

string → array of chars

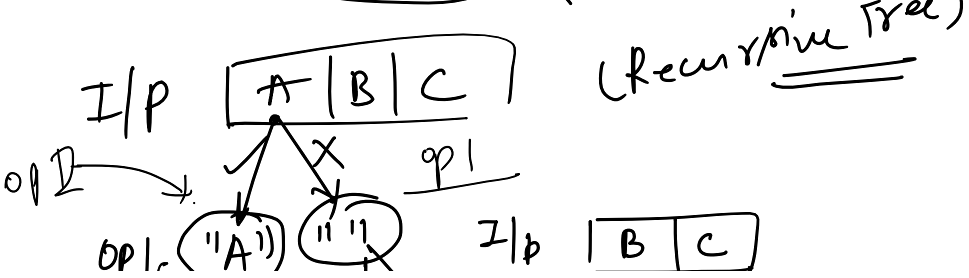
{ '~~[illegible]~~', 'A', 'B', 'C', "AB", "BC", "CA", "ABC" }

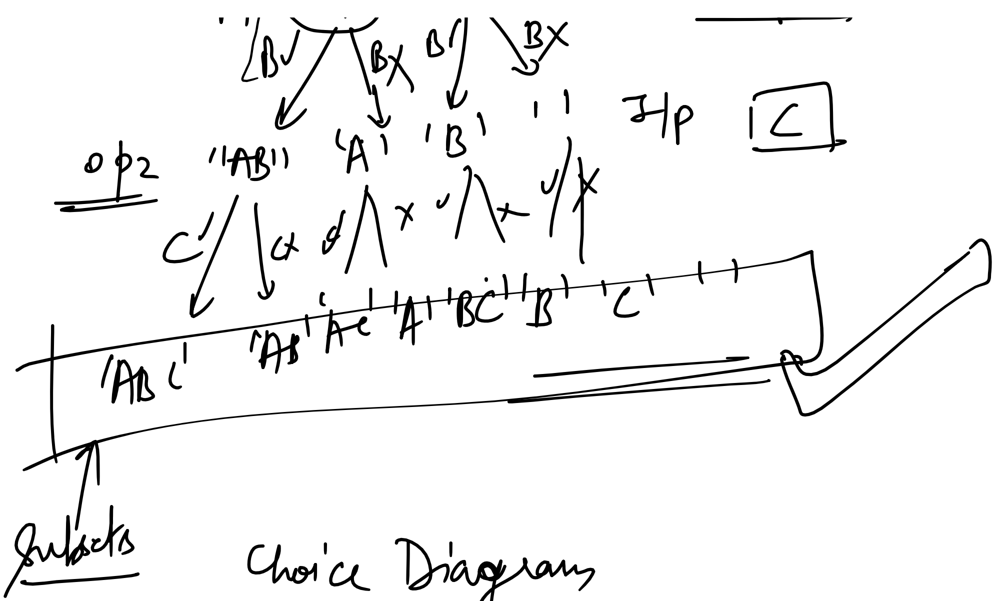
अगर कोई Choices given है और उन Choices से related कोई Decision लेने है तो तो Recursion से ही solve होगा ||



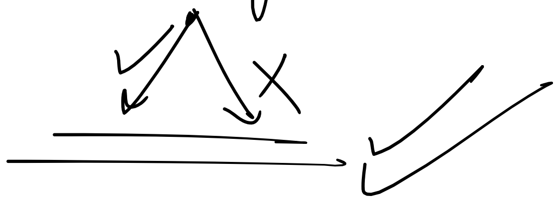
{ 'ABC', 'AB', 'AC', 'A', 'BC', 'B', 'C', '' }

Recursion Choice



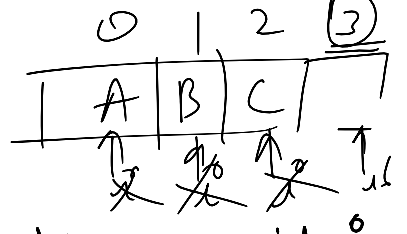


Choice Diagram



[string] o/p

```
vector<string> func1 (string & ip) {
        vector<string> ans;
        string temp1; //
        solve(ans, temp1, 0, ip);
        return ans;
}
```



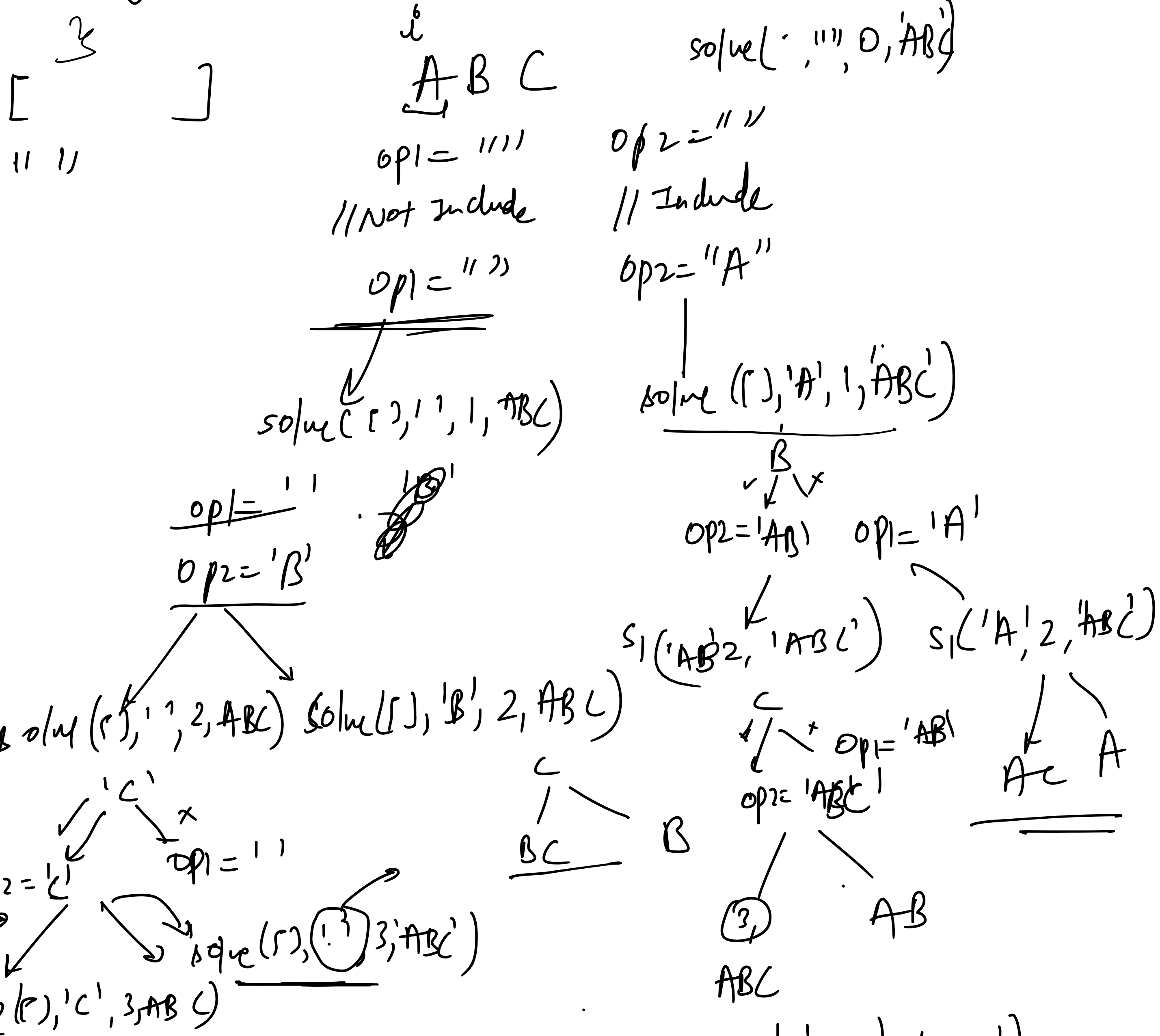
```
void solve(vector<string>&ans, string temp, int i,
            string ip) {
        if (i == ip.size()) {
                ans.push_back(temp); return;
        }
```

Choice Diagram

[illegible]

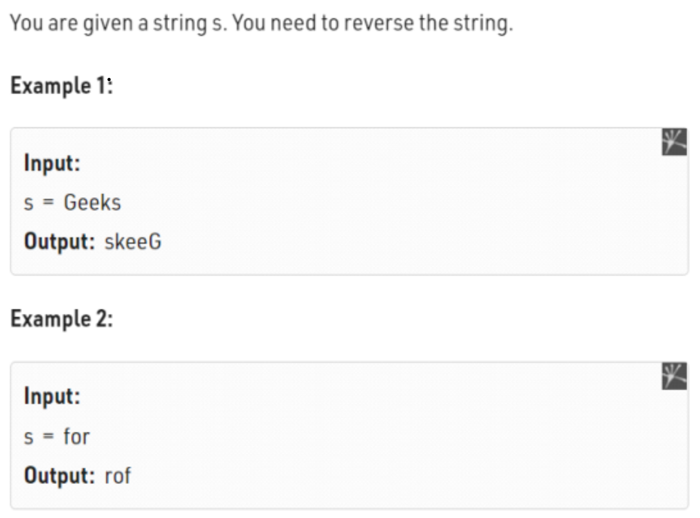
Choice Diagram

```
string op1 = temp;   // Not Include
string op2 = temp;   //Include
op2 += ip[i];
solve(ans, op1, i+1, ip);
solve( ans, op2, i+1, ip);
}
```



['C', ' ', 'BC', 'B', 'A', 'AB', 'AC', 'ABC']

All subsets (leetcode) → ✓
(GFG) → ✓



You are given a string s. You need to reverse the string.

Example 1:

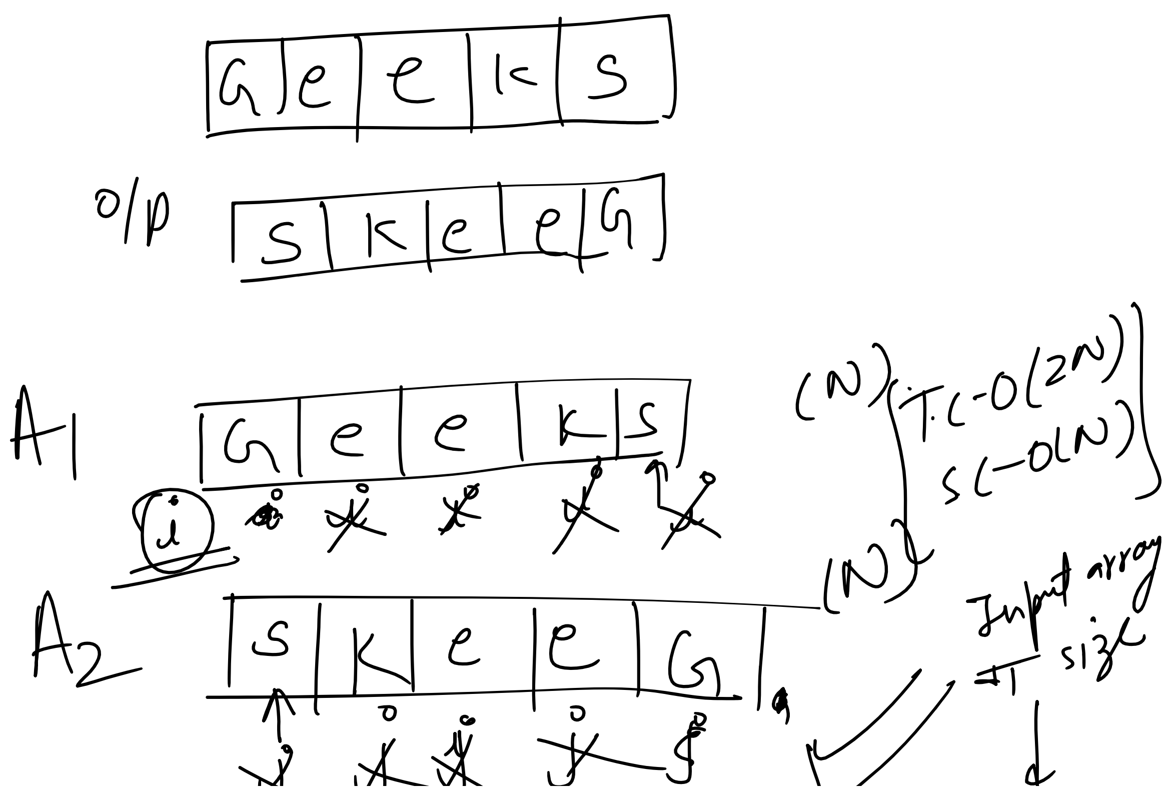
Input:
s = Geeks
Output: skeeG

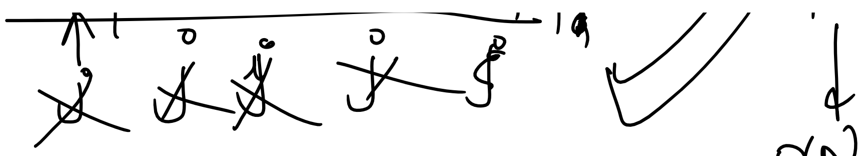
Example 2:

Input:
s = for
Output: rof

I/P Geeks

O/P skeeG





T.C — O(N)

O/p

s	k	e	e	G

```
string A2(A1.size());        ★ Space
 int i = A1.size()-1, j=0;
 while(i>=0){
     A2[j] = A1[i];
     j++;   i--;
 }
```

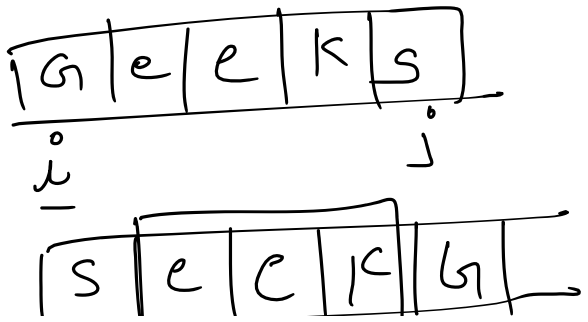
Time

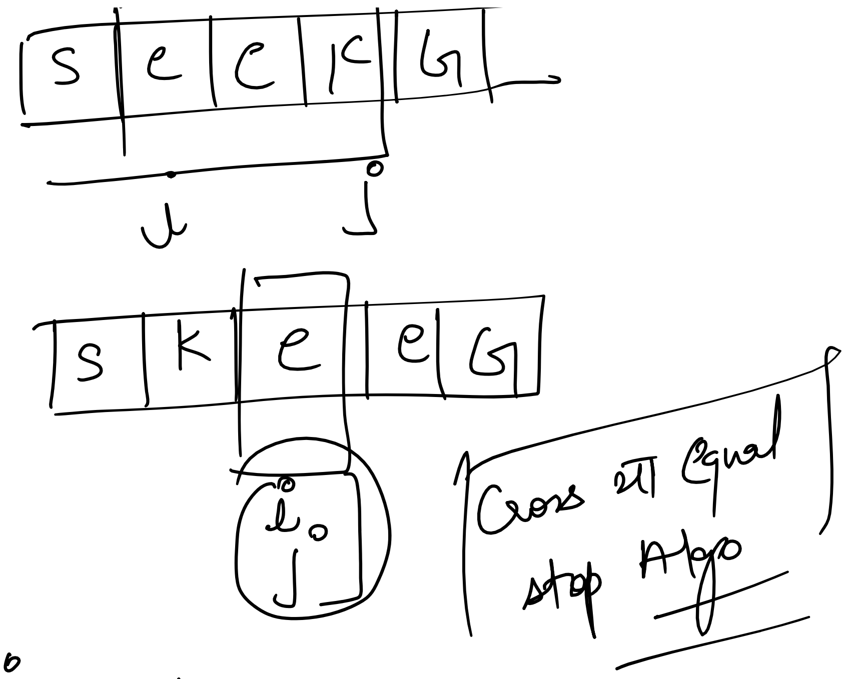
T.C - O(N)
S.C — O(N)

★★★★ 2 pointer Approach

G	e	e	k	s

steps
1. 1-point. (starting index)
2. 2-pointer (ending index)

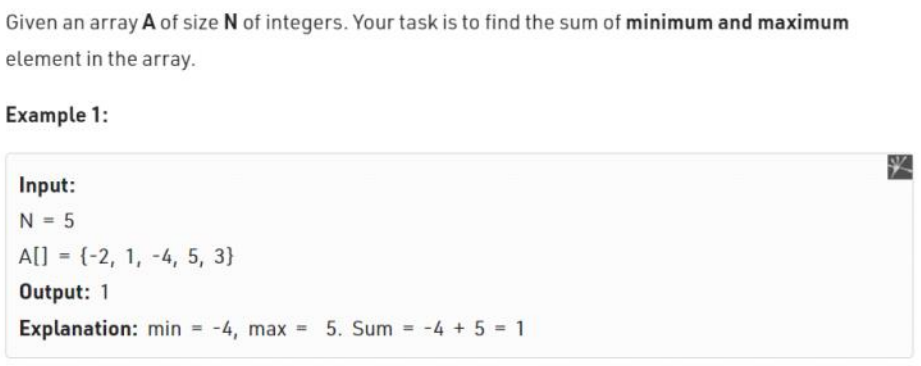




```
int i=0, j=n-1;
    while( i<j ) {
① ** swap(A[i], A[j]);
② **  { char temp = A[i];
          A[i] = A[j];
          A[j] = temp }
        j--; i++;
}
```

T.C - O(N)

S.C - O(1)


Given an array **A** of size **N** of integers. Your task is to find the sum of **minimum and maximum** element in the array.

Example 1:

Input:
N = 5
A[] = {-2, 1, -4, 5, 3}
Output: 1
Explanation: min = -4, max = 5. Sum = -4 + 5 = 1

| -2 | 1 | -4 | 5 | 3 |

maximum = 5
minimum = -4] $\boxed{2}$ $\underline{\text{Ans}}$ ↑

| -2 | 1 | -4 | 5 | 3 |

maximum

maximum = 5

```
int maximum = INT_MIN;
   for(int i=0; i<n; i++){
        if ( A[i] > maximum){
             maximum = A[i];
   }    }
```

= $\underline{\textcircled{1}}$

```
int minimum = INT_MAX;
   for(int i=0; i<n; i++){
        if(A[i] < minimum){
             minimum = A[i];
        }
   }
```

minimum = -4

T.C — O(N)

S.C — O(1)

| -2 | 1 | -4 | 5 | 3 |

-2	1	-4	5	3
	↑			

mini = -2
max = -2

current element मिलेगा वो अगर हमारे current minimum से छोटा होगा तो वो maximum तो कभी हो ही नहीं सकता

```
int minimum = A[0],
        maximum = A[0];

for(int i=1; i<n ; i++){
    if(A[i] < minimum){
        minimum = A[i];
        continue;
    }
    if(A[i] > maximum){
        maximum = A[i];
    }
}
```

-2	1	-4	5	3
	↑	~~↑~~	~~↑~~	↑

mini = -4
max = ~~-2~~ ~~1~~ 5

5 - 4 = 1 ✓

T. C - O(n)
S. C - O(1)