# Lesson

#include <iostream>

#include <stdlib.h>

#include <sstream>

#include <stack>

#include <string>

#include <list>

#include <vector>

using namespace std;

#include <iostream>

using namespace std;

class *Node*{

public:

*Node*\* lchild;

    int data;

*Node*\* rchild;

};

class *Queue*{

private:

    int size;

    int front;

    int rear;

*Node*\*\* Q;  // [Node\*]\*: Pointer to [Pointer to Node]

public:

    Queue(int *size*);

    ~Queue();

    bool isFull();

    bool isEmpty();

    void enqueue(*Node*\* *x*);

*Node*\* dequeue();

};

*Queue*::Queue(int *size*) {

*this*->size = *size*;

    front = -1;

    rear = -1;

    Q = **new** *Node*\* [*size*];

}

*Queue*::~Queue() {

**delete []** Q;

}

bool *Queue*::isEmpty() {

    if (front == rear){

        return true;

    }

    return false;

}

bool *Queue*::isFull() {

    if (rear == size-1){

        return true;

    }

    return false;

}

void *Queue*::enqueue(*Node*\* *x*) {

    if (isFull()){

        cout << "Queue Overflow" << endl;

    } else {

        rear++;

        Q[rear] = *x*;

    }

}

*Node*\* *Queue*::dequeue() {

*Node*\* x = nullptr;

    if (isEmpty()){

        cout << "Queue Underflow" << endl;

    } else {

        front++;

        x = Q[front];

    }

    return x;

}

*Node*\* root = **new** *Node*;

void createTree(){

*Node*\* p;

*Node*\* t;

    int x;

*Queue* q(10);

    cout << "Enter root value: " << flush;

    cin >> x;

    root->data = x;

    root->lchild = nullptr;

    root->rchild = nullptr;

    q.enqueue(root);

    while (! q.isEmpty()){

        p = q.dequeue();

        cout << "Enter left child value of " << p->data << ": " << flush;

        cin >> x;

        if (x != -1){

            t = **new** *Node*;

            t->data = x;

            t->lchild = nullptr;

            t->rchild = nullptr;

            p->lchild = t;

            q.enqueue(t);

        }

        cout << "Enter left child value of " << p->data << ": " << flush;

        cin >> x;

        if (x != -1){

            t = **new** *Node*;

            t->data = x;

            t->lchild = nullptr;

            t->rchild = nullptr;

            p->rchild = t;

            q.enqueue(t);

        }

    }

}

void preorder(*Node*\* *p*){

    if (*p*){

        cout << *p*->data << ", " << flush;

        preorder(*p*->lchild);

        preorder(*p*->rchild);

    }

}

void inorder(*Node*\* *p*){

    if (*p*){

        inorder(*p*->lchild);

        cout << *p*->data << ", " << flush;

        inorder(*p*->rchild);

    }

}

void postorder(*Node*\* *p*){

    if (*p*){

        postorder(*p*->lchild);

        postorder(*p*->rchild);

        cout << *p*->data << ", " << flush;

    }

}

int main() {

    createTree();

    preorder(root);

    cout << endl;

    inorder(root);

    cout << endl;

    postorder(root);

    cout << endl;

    return 0;

}

# Lab stack

## Q1

Implement all methods in class **Stack** with template type **T**. The description of each method is written as comment in frame code.

#ifndef STACK\_H

#define STACK\_H  
#include "DLinkedList.h"  
template<class T>  
class Stack {  
protected:  
    DLinkedList<T> list;  
public:  
    Stack() {}  
    void push(T item) ;  
    T pop() ;  
    T top() ;  
    bool empty() ;  
    int size() ;  
    void clear() ;  
};  
  
#endif

You can use all methods in class **DLinkedList**without implementing them again. The description of class **DLinkedList**is written as comment in frame code.

template <class T>  
class DLinkedList   
{  
public:  
    class Node;     //forward declaration  
protected:  
    Node\* head;  
    Node\* tail;  
    int count;  
public:  
    DLinkedList() ;  
    ~DLinkedList();  
    void add(const T& e);  
    void add(int index, const T& e);  
    T removeAt(int index);  
    bool removeItem(const T& removeItem);  
    bool empty();  
    int size();  
    void clear();  
    T get(int index);  
    void set(int index, const T& e);  
    int indexOf(const T& item);  
    bool contains(const T& item);  
};

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| Stack<int> stack;  cout << stack.empty() << " " << stack.size(); | 1 0 |
| Stack<int> stack;  int item[] = { 3, 1, 4, 5, 2, 8, 10, 12 };  for (int idx = 0; idx < 8; idx++) stack.push(item[idx]);    assert(stack.top() == 12);    stack.pop();  stack.pop();    cout << stack.top(); | 8 |

void push(T item) {

// TODO: Push new element into the top of the stack

}

T pop() {

// TODO: Remove an element on top of the stack

}

T top() {

// TODO: Get value of the element on top of the stack

}

bool empty() {

// TODO: Determine if the stack is empty

}

int size() {

// TODO: Get the size of the stack

}

void clear() {

// TODO: Clear all elements of the stack

}

void push(*T* *item*) {

    // TODO: Push new element into the top of the stack

        //default: push new element into the head of the DLL

        list.add(0,*item*);

    }

*T* pop() {

        // TODO: Remove an element on top of the stack

        if(list.empty()) cout<<"stack is empty"<<endl;

        //else

*T* temp = list.get(0);

        list.removeAt(0);

        return temp;

    }

*T* top() {

        // TODO: Get value of the element on top of the stack

        return list.get(0);

    }

    bool empty() {

        // TODO: Determine if the stack is empty

        return list.empty();

    }

    int size() {

        // TODO: Get the size of the stack

        return list.size();

    }

    void clear() {

        // TODO: Clear all elements of the stack

        list.clear();

    }

## Q2

Given an array nums[] of size N having distinct elements, the task is to find the next greater element for each element of the array

Next greater element of an element in the array is the nearest element on the right which is greater than the current element.

If there does not exist a next greater of a element, the next greater element for it is -1

Note: iostream, stack and vector are already included

Constraints:

1 <= nums.length <= 10^5

0 <= nums[i] <= 10^9

Example 1:

Input:

nums = {15, 2, 4, 10}

Output:

{-1, 4, 10, -1}

Example 2:

Input:

nums = {1, 4, 6, 9, 6}

Output:

{4, 6, 9, -1, -1}

For example:

|  |  |  |
| --- | --- | --- |
| **Test** | **Input** | **Result** |
| int N;  cin >> N;  vector<int> nums(N);  for(int i = 0; i < N; i++) cin >> nums[i];  vector<int> greaterNums = nextGreater(nums);  for(int i : greaterNums)  cout << i << ' ';  cout << '\n'; | 4  15 2 4 10 | -1 4 10 -1 |
| int N;  cin >> N;  vector<int> nums(N);  for(int i = 0; i < N; i++) cin >> nums[i];  vector<int> greaterNums = nextGreater(nums);  for(int i : greaterNums)  cout << i << ' ';  cout << '\n'; | 5  1 4 6 9 6 | 4 6 9 -1 -1 |

// iostream, stack and vector are included

vector<int> nextGreater(vector<int>& arr){

}

// iostream, stack and vector are included

//we dont need to include stack

//int: max / min = +- 2147483647>10^9

//# of for loop interations is not limited

vector<int> nextGreater(vector<int>& *arr*){

    int size = *arr*.size();

    for(int i=0; i<size;i++){

        int temp = *arr*[i];

        for(int j=i; j<size;j++){

            if(*arr*[i] < *arr*[j]) {

*arr*[i] = *arr*[j];

                break;

            }

        }

        if(*arr*[i] == temp) *arr*[i] = -1;

    }

    return *arr*;

}

## Q3

Given a string consisting of 'a'-'z' and brackets '(' and ')'.

Your task is to implement a function with following prototype:

string parenthesesReversal(string s);

The function returns the string after recursively reversing every string enclosed between brackets.

**Note:**

- The iostream, vector and stack libraries have been included and namespace std is being used. No other libraries are allowed.

- You can write helper functions.

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| string s = "vi(ik)kq";  cout << parenthesesReversal(s); | vikikq |

//reference: https://leetcode.com/problems/reverse-substrings-between-each-pair-of-parentheses/solutions/383670/java-c-python-tenet-o-n-solution/

*string* parenthesesReversal(*string* *s*) {

    // STUDENT ANSWER

    int n = *s*.length();

        vector<int> opened, pair(n);

        for (int i = 0; i < n; ++i) {

            if (*s*[i] == '(')

                opened.push\_back(i);

            if (*s*[i] == ')') {

                int j = opened.back();

                opened.pop\_back();

                pair[i] = j;

                pair[j] = i;

            }

        }

*string* res;

        for (int i = 0, d = 1; i < n; i += d) {

            if (*s*[i] == '(' || *s*[i] == ')')

                i = pair[i], d = -d;

            else

                res += *s*[i];

        }

        return res;

}

## Q4

Given a string **S** of characters, a *duplicate removal* consists of choosing two adjacent and equal letters, and removing them.

We repeatedly make duplicate removals on **S** until we no longer can.

Return the final string after all such duplicate removals have been made.

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| cout << removeDuplicates("abbaca"); | ca |
| cout << removeDuplicates("aab"); | b |

*string* removeDuplicates(*string* *S*){

    /\*TODO\*/

    if(*S* == "") return "";

    stack<char> st;

    int i = 0;

    st.push(*S*[i]);

    i++;

    int size = *S*.length(); //error: comparison between signed and unsigned integer expressions

    while(i < size){

        if(st.empty() || st.top() != *S*[i]){

            st.push(*S*[i]);

            i++;

        }

        else{

            st.pop();

            i++;

        }

    }

*string* res = "";

    while(!st.empty()){

        res = st.top() + res;

        //perfect, we dont need to reverse the string

        st.pop();

    }

    return res;

}

## Q5

Given a string s and an integer k (k >= 1). Manipulate the string following these rules:

- Repeatly delete the earliest k-consecutive characters appear in string s.

- There is no k-consecutive characters in the final string (after manipulating).

**Request:** Implement function

string deleteDuplicate(string s, int k);

To return the final string.

Example:

- deleteDuplicate("aabbbdddadd", 3).

- The final string returned is dd.

- Explanation: Firstly, delete the "bbb" to get "aadddadd", then delete "ddd" to get "aaadd", then delete "aaa" and final string is "dd".

*Note: In this exercise, libraries iostream, stack, utility and using namespace std have been used. You can add other functions; however, you are not allowed to add other libraries.*

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| string s = "aabbbdddadd";  int k = 3;  cout << deleteDuplicate(s, k); | dd |
| string s = "aaabbbbccddddefffghiiiiihhhhggggffda";  int k = 5;  cout << deleteDuplicate(s, k); | aaabbbbccddddeda |

string deleteDuplicate(string s, int k) {

}

using namespace std;

int startIndex(*string* *s*, int *k*){

    int size = *s*.length();

    for(int i=0; i<size; i++){

        bool equal = true;

        for(int j=i; j<i+*k*-1; j++){

            if(*s*[j] != *s*[j+1]) equal = false;

        }

        if(equal == true) return i;

    }

    return -1;

}

*string* deleteDuplicate(*string* *s*, int *k*) {

    int idx = startIndex(*s*,*k*);

    //base case

    if(idx == -1) return *s*;

    //normal case

    if(idx == 0){

        //index: [idx + k, end]

        int count = (*s*.length()-1) - (idx+*k*) + 1;

        //string ( other\_string, position, count ) - no need include <string>`

*string* res(*s*,idx + *k*,count);

        //recursion

        return deleteDuplicate(res,*k*);

    }

    else{

        //index: [0,idx-1]

        //index: [idx + k, end]

        int count1 = idx;

        int count2 = (*s*.length()-1) - (idx + *k*) + 1;

*string* res1(*s*,0,count1);

*string* res2(*s*,idx+*k*,count2);

        //recursion

        return deleteDuplicate(res1+res2,*k*);

    }

}

## Q6 – dung mot phan (time limit exceed)

Given an array of integers.

Your task is to implement a function with following prototype:

int sumOfMinSubarray(vector<int>& nums);

The function returns the sum of the minimum value of every subarray of nums. The sum may be too large, so the result should mod 10000.

**Note:**

- The iostream, vector and stack libraries have been included and namespace std is being used. No other libraries are allowed.

- You can write helper functions.

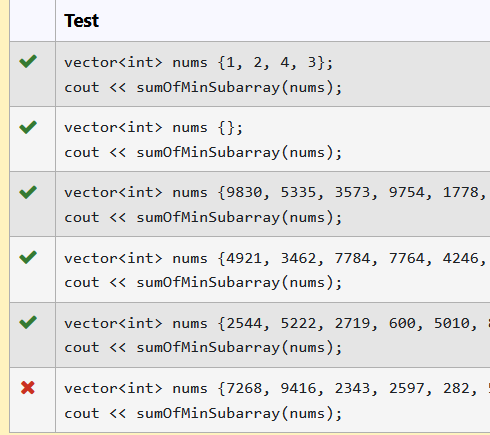
For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| vector<int> nums {1, 2, 4, 3};  cout << sumOfMinSubarray(nums); | 20 |

int sumOfMinSubarray(vector<int>& nums) {

// STUDENT ANSWER

}



int minOfSt(stack<int> *st*){

    int tmp=*st*.top();

    while(!*st*.empty()){

        if(*st*.top() < tmp) tmp = *st*.top();

*st*.pop();

    }

    return tmp;

}

int sumOfMinSubarray(vector<int>& *nums*) {

    // STUDENT ANSWER

    //example: vector<int>: 1 2 4 3

    int res = 0;

    int size = *nums*.size(); // convert unsigned int to unsigned int (vector size)

    for(int i = 0; i < size; i++){

        //value of i: create stacks from position i

        //4 stacks

        stack<int> st;

        for(int j=i; j < size; j++) st.push(*nums*[j]);

        //stack 1: 1 2 4 3

        //stack 2: 2 4 3

        //stack 3: 4 3

        //stack 4: 3

        while(!st.empty()){

            res += minOfSt(st);

            st.pop();

        }

    }

    return res%10000;

}

### Code tham khao

int sumOfMinSubarray(vector<int>& a) {

// STUDENT ANSWER

int mod = 10000;

stack<int>st;

int n = a.size();

int dp[n];

for (int i =0; i<n; i++)

{

while (!st.empty()&&a[st.top()]>=a[i])

{

st.pop();

}

if (!st.empty())

{

int pre = st.top();

dp[i] = dp[pre]+(i-pre)\*a[i];

}

else {

// or it doesn't exist, in this case the current element

// contributes with all subarrays ending at i

dp[i] = (i + 1) \* a[i];

}

// push the current index

st.push(i);

}

int res = 0;

for (int i =0; i<n; i++)

{

res+=dp[i];

res%=mod;

}

return res;

}

## Q7

Given a string **s** containing just the characters **'(', ')', '[', ']', '{', and '}'**. Check if the input string is valid based on following rules:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

For example:

* String **"[]()"** is a valid string, also **"[()]".**
* String "**[])**" is **not** a valid string.

Your task is to implement the function

bool isValidParentheses (string s){  
    /\*TODO\*/  
}

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| cout << isValidParentheses("[]"); | 1 |
| cout << isValidParentheses("[]()"); | 1 |
| cout << isValidParentheses("[)"); | 0 |

bool isValidParentheses (string s){

/\*TODO\*/

}

bool isValidParentheses (*string* *s*){

    /\*TODO\*/

    /\*

    ascii table

    40 (

    41 )

    91 [

    93 ]

    123 {

    125 }

    \*/

    int size = *s*.length();

    stack<char> st;

    for(int i = 0; i < size;i++){

        if(*s*[i] == '(' || *s*[i] == '{' || *s*[i] == '['){

            st.push(*s*[i]);

        }

        else if(*s*[i] == ')' || *s*[i] == '}' || *s*[i] == ']'){

            if(st.empty()) return false;

            //else: st is not empty

            char x = st.top();

            if(*s*[i] == ')'){

                //ascii: ) < 50, } and ] > 50

                if(*s*[i] - x != 1) return false;

                else st.pop();

            }

            else{

                if(*s*[i] - x != 2) return false;

                else st.pop();

            }

        }

        //else: do nothing, next

    }

    if(st.empty()) return true;

    else return false;

}

# Lab queue

## Q8

Implement all methods in class **Queue** with template type **T**. The description of each method is written as comment in frame code.

#ifndef QUEUE\_H

#define QUEUE\_H  
#include "DLinkedList.h"  
template<class T>  
class Queue {  
protected:  
    DLinkedList<T> list;  
public:  
    Queue() {}  
    void push(T item) ;  
    T pop() ;  
    T top() ;  
    bool empty() ;  
    int size() ;  
    void clear() ;  
};  
  
#endif /\* QUEUE\_H \*/

You can use all methods in class **DLinkedList** without implementing them again. The description of class **DLinkedList** is written as comment in frame code.

template <class T>  
class DLinkedList   
{  
public:  
    class Node;     //forward declaration  
protected:  
    Node\* head;  
    Node\* tail;  
    int count;  
public:  
    DLinkedList() ;  
    ~DLinkedList();  
    void add(const T& e);  
    void add(int index, const T& e);  
    T removeAt(int index);  
    bool removeItem(const T& removeItem);  
    bool empty();  
    int size();  
    void clear();  
    T get(int index);  
    void set(int index, const T& e);  
    int indexOf(const T& item);  
    bool contains(const T& item);  
};

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| Queue<int> queue;  assert(queue.empty());  assert(queue.size() == 0); |  |

void push(*T* *item*) {

    // TODO: Push new element into the end of the queue

*list*.add(*item*);

}

*T* pop() {

    // TODO: Remove an element in the head of the queue

    if(*list*.size() != 0){

        T res = list.get(0);

*list*.removeAt(0);

        return res;

    }

    throw *out\_of\_range*("queue is empty");

}

*T* top() {

    // TODO: Get value of the element in the head of the queue

    if(*list*.size() != 0){

        T res = list.get(0);

        return res;

    }

    throw *out\_of\_range*("queue is empty");

}

bool empty() {

    // TODO: Determine if the queue is empty

    return list.empty();

}

int size() {

    // TODO: Get the size of the queue

    return list.size();

}

void clear() {

    // TODO: Clear all elements of the queue

*list*.clear();

}

## Q9

A nice number is a positive integer that contains only 2's and 5's.

Some nice numbers are: 2, 5, 22, 25, 52, 55, ...

Number 2 is the first nice number.

Given an integer N, return the Nth nice number.

Note: iostream, vector, queue are already included for you.

Constraint:

1 <= n <= 10^6

Example 1:

Input:

n = 5

Output:

52

Explanation:

The sequence of nice numbers is 2, 5, 22, 25, 52, 55, ...

The 5th number in this sequence is 52

Example 2:

Input:

n = 10000

Output:

2255522252225

For example:

|  |  |  |
| --- | --- | --- |
| **Test** | **Input** | **Result** |
| int n;  cin >> n;  cout << nthNiceNumber(n) << endl; | 5 | 52 |
| int n;  cin >> n;  cout << nthNiceNumber(n) << endl; | 10000 | 2255522252225 |

// iostream, vector and queue are included

// You can write helper methods

long long nthNiceNumber(int *n*) {

    long long \*nice = **new** long long[*n*+1];

    nice[0] = 0;

    nice[1] = 2;

    nice[2] = 5;

    for(int i = 3; i < *n*+1; i++){

        if(i%2 != 0)   nice[i] = nice[(i-1)/2] \* 10 + 2;

        else nice[i] = nice[(i-1)/2] \* 10 + 5;

    }

    return nice[*n*];

}

Use idea HEAP

0 [0]

2 [1]

5 [2]

22 [3]

25 [4]

52

55

222

225

252

255

522

525

Node at index: i

Parent: underbounded of (i-1)/2

Children = parent\*10 + 2 or parent\*10 + 5

## Q10

A group of N students in HCMUT are playing a funny game. They gather around a circle and number themselves from 1 to N clockwise. After a step of the game, a person is removed from the circle. The last person to stay in the circle is the winner.

The game's rule is as follows:

1. The game start at the person numbered 1.

2. From the current person, count k people clockwise (including the person you started at). The counting may wraps around the circle.

3. The last counted one is remove from the circle.

4. If the circle still has more than one people, the game continues from the person immediately clockwise of the person who just lost the game. Then repeat step 2

5. The last person in the game will win.

Toan really wants to win the game to impress their friends. Given the number of players, N, and an integer, k. Help Toan win the game by determine the number in which he has to be standing to certainly win the game.

Hint: You can use a queue to simulate the process of the game.

Constraint:

1 <= k <= N <= 10^4

Example:

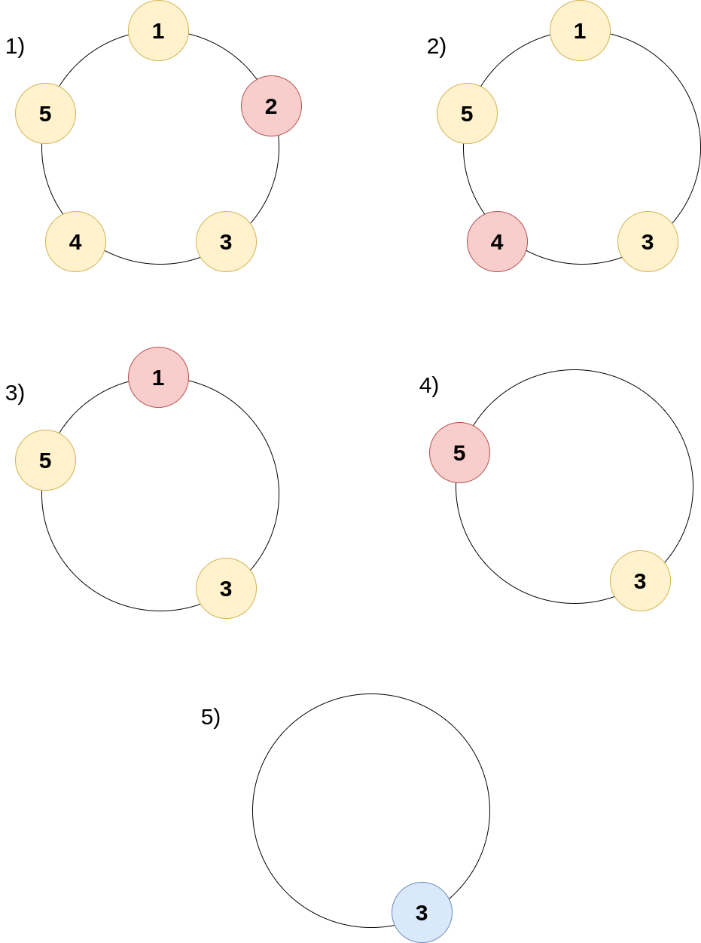
Input:

n = 5, k = 2

Output

3

Explanation: The game proceeds as follows



For example:

|  |  |  |
| --- | --- | --- |
| **Test** | **Input** | **Result** |
| int N;  int k;  cin >> N >> k;  cout << numberOfTheWinner(N, k); | 5  2 | 3 |
| int N;  int k;  cin >> N >> k;  cout << numberOfTheWinner(N, k); | 6  5 | 1 |

// iostream and queue are included

// Hint: Use a queue to simulate the process

int numberOfTheWinner(int *N*, int *k*) {

    queue<int> pos;

    for(int i = 1; i <= *N*; i++) pos.push(i);

    while(pos.size() != 1){

        //loop k-1 times: get the front and push it back

        for(int i = 0; i < *k*-1; i++){

            int tmp = pos.front();

            pos.pop();

            pos.push(tmp);

        }

        pos.pop(); //result: one position is deleted

    }

    return pos.front();

}

## Q11 – dung mot phan (time limit exceed)

Given an array of integers.

Your task is to implement a function with following prototype:

int sumOfMaxSubarray(vector<int>& nums, int k);

The function returns the sum of the maximum value of every subarray of nums with fixed length k.

**Note:**

- The iostream, vector, queue and deque libraries have been included and namespace std is being used. No other libraries are allowed.

- You can write helper functions and classes.

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| vector<int> nums {1, 2, 4, 3, 6};  int k = 3;  cout << sumOfMaxSubarray(nums, k); | 14 |

Explain the task:

Subarr 1: 1 2 4 -> max = 4

Subarr 2: 2 4 3 -> max = 4

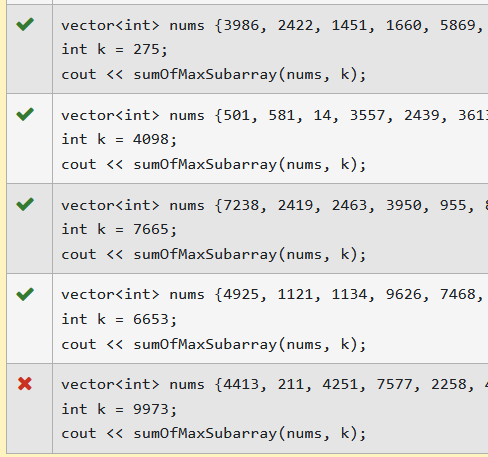
Subarr 3: 4 3 6 -> max = 6

Result = 4+4+6 = 14

int sumOfMaxSubarray(vector<int>& nums, int k) {

// STUDENT ANSWER

}



int maxOfQueue(queue<int> *q*){

    int temp = *q*.front();

    while(*q*.size()){

        if(temp < *q*.front()) temp = *q*.front();

*q*.pop();

    }

    return temp;

}

int sumOfMaxSubarray(vector<int>& *nums*, int *k*) {

    // STUDENT ANSWER

    //base case

    if(*k*<=0) return 0;

    int res = 0;

    int pos = 0;

    queue<int> q;

    q.push(0);// virtual push

    for(int i=0; i<*k*-1; i++){

        //push k-1 values to queue

        q.push(*nums*[pos++]);

    }

    int size = *nums*.size();

    while(pos < size){

        q.pop();

        q.push(*nums*[pos++]);

        res += maxOfQueue(q);

    }

    return res;

}

### Code tham khảo

int sumOfMaxSubarray(vector<int>& nums, int k) {

deque<int> d; // deque to store INDICES of Maximums (usually in the front of deque)

// deque elements would be arranged:

// Max --- Min

int sum = 0;

// Push in maximum of the first subarray

int i = 0;

while (i < k)

{

while (!d.empty() && nums.at(i) >= nums.at(d.back()))

d.pop\_back();

d.push\_back(i);

++i;

}

sum += nums.at(d.front()); // Front is always a maximum value of the current subarray

// Continue with other elements

// nums[i] is new element added

while (i < (int)nums.size())

{

// Pop old elements

if (!d.empty() && d.front() < i - k + 1)

d.pop\_front();

while (!d.empty() && nums.at(i) >= nums.at(d.back()))

d.pop\_back();

d.push\_back(i);

++i;

sum += nums.at(d.front());

}

return sum;

}

# Lab sort

## Q12

Implement method bubbleSort() in class SLinkedList to sort this list in ascending order. After each bubble, we will print out a list to check (using printList).

#include <iostream>

#include <sstream>

using namespace std;

template <class T>

class SLinkedList {

public:

class Node; // Forward declaration

protected:

Node\* head;

Node\* tail;

int count;

public:

SLinkedList()

{

this->head = nullptr;

this->tail = nullptr;

this->count = 0;

}

~SLinkedList(){};

void add(T e)

{

Node \*pNew = new Node(e);

if (this->count == 0)

{

this->head = this->tail = pNew;

}

else

{

this->tail->next = pNew;

this->tail = pNew;

}

this->count++;

}

int size()

{

return this->count;

}

void printList()

{

stringstream ss;

ss << "[";

Node \*ptr = head;

while (ptr != tail)

{

ss << ptr->data << ",";

ptr = ptr->next;

}

if (count > 0)

ss << ptr->data << "]";

else

ss << "]";

cout << ss.str() << endl;

}

public:

class Node {

private:

T data;

Node\* next;

friend class SLinkedList<T>;

public:

Node() {

next = 0;

}

Node(T data) {

this->data = data;

this->next = nullptr;

}

};

void bubbleSort();

};

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| int arr[] = {9, 2, 8, 4, 1};  SLinkedList<int> list;  for(int i = 0; i <int(sizeof(arr))/4;i++)  list.add(arr[i]);  list.bubbleSort(); | [2,8,4,1,9]  [2,4,1,8,9]  [2,1,4,8,9]  [1,2,4,8,9] |

template <class T>

void SLinkedList<T>::bubbleSort(){

}

template <class *T*>

void SLinkedList<*T*>::bubbleSort(){

    for(int i = 0; i <*this*->count -1; i++){

        Node\* nxt = head;

        while(nxt != tail){

            if(nxt->data > nxt->next->data) {

                T Ttmp = nxt->data;

                nxt->data = nxt->next->data;

                nxt->next->data = Ttmp;

            }

            nxt = nxt->next; //point to next

        }

        printList();

    }

}

## Q13

QuickSort is one of the fastest sorting algorithms for sorting large data. When implemented well, it can be about two or three times faster than its main competitors, such as MergeSort and HeapSort. When the number of elements is below some threshold (min\_size), switch to insertion sort - non-recursive sorting algorithm that performs fewer swaps, comparisons or other operations on such small arrays.

Implement static methods **hybridQuickSort** in class Sorting to sort an array in ascending order. If you do insertion sort, please print "Insertion sort: array" (using printArray) first. If you do quick sort, please print "Quick sort: array (using printArray)" first. Please read example carefully to know exactly what we print.

To match the test cases, please note:

* Using first element as pivot
* Recursively call the hybridQuickSort function to the left of the pivot first, then recursively to the right of the pivot
* Do insertion sort if sub array size smaller than min\_size

#include <iostream>

using namespace std;

template <class T>

class Sorting

{

private:

static T \*Partition(T \*start, T \*end);

public:

static void printArray(T \*start, T \*end)

{

int size = end - start;

for (int i = 0; i < size - 1; i++)

cout << start[i] << ", ";

cout << start[size - 1];

cout << endl;

}

static void insertionSort(T \*start, T \*end);

static void hybridQuickSort(T \*start, T \*end, int min\_size);

};

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| int array[] = {1, 2, 6, 4, 7, 8, 5, 3};  int min\_size = 4;  Sorting<int>::hybridQuickSort(&array[0], &array[8], min\_size); | Quick sort: 1, 2, 6, 4, 7, 8, 5, 3  Quick sort: 2, 6, 4, 7, 8, 5, 3  Quick sort: 6, 4, 7, 8, 5, 3  Insertion sort: 5, 4, 3  Insertion sort: 8, 7 |
| int array[] = {2, 6, 4, 12, 23, 1, 0, -12};  int min\_size = 4;  Sorting<int>::hybridQuickSort(&array[0], &array[8], min\_size); | Quick sort: 2, 6, 4, 12, 23, 1, 0, -12  Insertion sort: 1, -12, 0  Quick sort: 23, 12, 4, 6  Insertion sort: 6, 12, 4 |
| int array[] = {30, 7, 20, 0, -13, 1, 19, 72};  int min\_size = 3;  Sorting<int>::hybridQuickSort(&array[0], &array[8], min\_size); | Quick sort: 30, 7, 20, 0, -13, 1, 19, 72  Quick sort: 19, 7, 20, 0, -13, 1  Quick sort: -13, 7, 1, 0  Quick sort: 7, 1, 0  Insertion sort: 0, 1  Insertion sort: 20  Insertion sort: 72 |

template <class T>

T \*Sorting<T>::Partition(T \*start, T \*end){

}

template <class T>

void Sorting<T>::insertionSort(T \*start, T \*end){

}

template <class T>

void Sorting<T>::hybridQuickSort(T \*start, T \*end, int min\_size){

}

/\*how to show data (input is pointer)

    int\* a = &array[1];

    int\* b = &array[7];

    next value: \*(a+1)

    count # of elements: b-a+1

    \*/

template <class *T*>

void swap(*T*\* *ptr1*, *T*\* *ptr2*){

*T* temp = \**ptr1*;

    \**ptr1* = \**ptr2*;

    \**ptr2* = temp;

}

template <class *T*>

*T* \*Sorting<*T*>::Partition(*T* \**start*, *T* \**end*){

*T* pivot = \**start*;

*T*\* i = *start*;

*T*\* j = *end*;

    do{

        do{i++;} while(\*i <= pivot);

        while(\*j > pivot) {j--;}

        if(i<j) swap(i,j);

    } while(i<j);

    swap(*start*,j);

    return j;

}

template <class *T*>

void Sorting<*T*>::insertionSort(*T* \**start*, *T* \**end*){

    for(*T*\* i=*start*+1; i<=*end*; i++){

*T*\* j = i-1;

*T* x = \*i;

        while(j>= *start* && \*j > x){

            \*(j+1) = \*j;

            j--;

        }

        \*(j+1) = x;

    }

}

template <class *T*>

void Sorting<*T*>::hybridQuickSort(*T* \**start*, *T* \**end*, int *min\_size*){

    if(*start* < *end*){

        if(*end* - *start* < *min\_size*){

            cout<<"Insertion sort: ";

            printArray(*start*,*end*);

            insertionSort(*start*,*end*);

        }

        else{

            cout<<"Quick sort: ";

            printArray(*start*,*end*);

*T*\* j = Partition(*start*,*end*);

            hybridQuickSort(*start*,j,*min\_size*);

            hybridQuickSort(j+1,*end*,*min\_size*);

        }

    }

}

## Q14

<https://riptutorial.com/algorithm/example/24482/odd-even-sort-basic-information>

Implement static method oddEvenSortin class **Sorting**to sort an array in **ascending** order.  After each selection, we will print out a list to check (using printArray method). **Remember to sort even first**.

Remind about odd even sort:

* This algorithm is divided into two phases - odd and even Phase. The algorithm runs until the array elements are sorted and in each iteration two phases mentioned above occurs.
* In the odd phase, we perform a bubble sort on odd indexed elements and in the even phase, we perform a bubble sort on even indexed elements.

#include <iostream>

using namespace std;

template <class T>

class Sorting

{

public:

/\* Function to print an array \*/

static void printArray(T \*start, T \*end)

{

int size = end - start;

for (int i = 0; i < size - 1; i++)

cout << start[i] << ", ";

cout << start[size - 1];

cout << endl;

}

static void oddEvenSort(T \*start, T \*end);

};

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| int arr[] = {3, 2, 3, 8, 5, 6, 4, 1};  Sorting<int>::oddEvenSort(&arr[0], &arr[8]); | 2, 3, 3, 5, 8, 1, 6, 4  2, 3, 3, 1, 5, 4, 8, 6  2, 1, 3, 3, 4, 5, 6, 8  1, 2, 3, 3, 4, 5, 6, 8  1, 2, 3, 3, 4, 5, 6, 8 |

template <class T>

void Sorting<T>::oddEvenSort(T \*start, T \*end)

{}

void Sorting<*T*>::oddEvenSort(*T* \**start*, *T* \**end*)

{

    bool sorted=0;

    while(sorted==0){

        sorted=1;

        for(int i=0; i<=end-start-2; i+=2){

            if(\*(start+i) > \*(start+i+1)){

                T temp = \*(start+i);

                \*(start+i) = \*(start+i+1);

                \*(start+i+1) = temp;

                sorted = 0;

            }

        }

        for(int i=1;i<=end-start-2;i+=2){

            if(\*(start+i) > \*(start+i+1)){

                T temp = \*(start+i);

                \*(start+i) = \*(start+i+1);

                \*(start+i+1) = temp;

                sorted = 0;

            }

        }

        printArray(start,end);

    }

}

## Q15 - redo

Implement static methods **sortSegment** and **ShellSort**in class **Sorting**to sort an array in ascending order.

#ifndef SORTING\_H

#define SORTING\_H

#include <sstream>

#include <iostream>

#include <type\_traits>

using namespace std;

template <class T>

class Sorting {

private:

static void printArray(T\* start, T\* end)

{

int size = end - start;

for (int i = 0; i < size; i++)

cout << start[i] << " ";

cout << endl;

}

static void sortSegment(T\* start, T\* end, int segment\_idx, int cur\_segment\_total) ;

public:

static void ShellSort(T\* start, T\* end, int\* num\_segment\_list, int num\_phases) ;

};

#endif /\* SORTING\_H \*/

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| int num\_segment\_list[] = {1, 3, 5};  int num\_phases = 3;  int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 };  Sorting<int>::ShellSort(&array[0], &array[10], &num\_segment\_list[0], num\_phases); | 5 segments: 5 4 3 2 1 10 9 8 7 6  3 segments: 2 1 3 5 4 7 6 8 10 9  1 segments: 1 2 3 4 5 6 7 8 9 10 |

static void sortSegment(T\* start, T\* end, int segment\_idx, int cur\_segment\_total) {

// TODO

}

static void ShellSort(T\* start, T\* end, int\* num\_segment\_list, int num\_phases) {

// TODO

// Note: You must print out the array after sorting segments to check whether your algorithm is true.

}

static void sortSegment(*T*\* *start*, *T*\* *end*, int *segment\_idx*, int *cur\_segment\_total*) {

    // TODO

    int size = end - start;

    for (int curr = segment\_idx + cur\_segment\_total; curr < size; curr += cur\_segment\_total) {

        int tmp = start[curr];

        int i;

        for (i = curr - cur\_segment\_total; i >= 0 && start[i] > tmp;

            i -= cur\_segment\_total) {

                start[i + cur\_segment\_total] = start[i];

            }

        start[i + cur\_segment\_total] = tmp;

    }

}

static void ShellSort(*T*\* *start*, *T*\* *end*, int\* *num\_segment\_list*, int *num\_phases*) {

    // TODO

    // Note: You must print out the array after sorting segments to check whether your algorithm is true.

    for (int phase = num\_phases - 1; phase >= 0; phase--)

    {

        int step = num\_segment\_list[phase]; //gap or segment

        for (int segment = 0; segment < step; segment++)

        {

            sortSegment(start, end, segment, step);

        }

        cout << step << " segments: ";

        printArray(start, end);

    }

}

## Q16

Implement static method selectionSortin class **Sorting**to sort an array in ascending order.  After each selection, we will print out a list to check (using printArray).

#include <iostream>

using namespace std;

template <class T>

class Sorting

{

public:

/\* Function to print an array \*/

static void printArray(T \*start, T \*end)

{

int size = end - start;

for (int i = 0; i < size - 1; i++)

cout << start[i] << ", ";

cout << start[size - 1];

cout << endl;

}

static void selectionSort(T \*start, T \*end);

};

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| int arr[] = {9, 2, 8, 1, 0, -2};  Sorting<int>::selectionSort(&arr[0], &arr[6]); | -2, 2, 8, 1, 0, 9  -2, 0, 8, 1, 2, 9  -2, 0, 1, 8, 2, 9  -2, 0, 1, 2, 8, 9  -2, 0, 1, 2, 8, 9 |

template <class T>

void Sorting<T>::selectionSort(T \*start, T \*end){}

template <class *T*>

void swap(*T* \**v1*, *T* \**v2*){

*T* tmp = \**v1*;

    \**v1* = \**v2*;

    \**v2* = tmp;

}

template <class *T*>

void Sorting<*T*>::selectionSort(*T* \**start*, *T* \**end*){

*T*\* i = *start*;

*T*\* j;

*T*\* k;

    while(i < *end* -1){

        j=i; k=i;

        while(j < *end*){

            if(\*j < \*k) k=j;

            j++;

        }

        swap(i,k);

        printArray(*start*,*end*);

        i++;

    }

}