**Binary seach //O(logn)**

int binarysearch(int a[], int n, int target) {

int L = 0, R = n – 1;

while( L < R) {

int mid = L + (R - L )/2;

if( a[mid] < target)

L = mid - 1;

else if (a[mid] > target)

R = mid - 1;

else

return mid;

}

return -1;

}

int binarysearch2(int a[], int n, int target) {

int L = 0, R = n – 1;

while( L +1 < R ) {

int mid = L + (R – L ) /2;

if( a[mid] < target)

L = mid;

else if ( a[mid] > target)

R = mid;

else

return mid;

}

if(a[L] == target) return L;

if(a[R] == target) return R;

return -1;

}

**countPrime //O(nlog(logn))**

int countPrime(int n) {

if(n < 2) return false;

vector<int> table(n, true);

table[0] = false; //1 is not prime

for(int i = 3; i < n; i+=2) //pass all even

table[i] == false;

for(inti = 3; i <= sqrt(n); i++) { //begin from prime 3

if(table[i - 1]) {

for(int j = i\*i; j < n; j += 2\*i)

table[j - 1] = false;

}

}

int count = 0;

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

if(table[i]) count++;

}

**gcd lcm //O(logn)**

int gcd(int a, int b)

return b == 0 ? a : gcd(b, a % b);

int lcm(int a, int b)

return a\* b/ gcd(a, b)

**power, powermod //O(logn)**

int power(double x, int n) {

double ans = 1;

while( n > 0) {

if( n % 2 != 0)

ans = ans \* x;

x = x \* x;

n = n / 2;

}

return ans;

}

int powermod(double x, int n, double m) {

double ans = 1;

while( n > 0) {

if( n % 2 != 0)

ans = ans \* x mod m;

x = x \* x mod m ;

n = n / 2;

}

return ans;

}

**Fermat //O(nlogn)**

bool Fermat(int p, int k) {

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

int a = rand() % (p-1) + 2;

if( powermod(2, p-1, p) != 1)

return false;

}

return true; (probably )

}

**Miller-Rabin //O(nlogn)**

bool Miller-Rabin(int n, int k) {

int s = 0;

while( ( (n-1) & (1<<s) ) != 1) {s++;}

double d = (n-1)/power(2,s);

for(inti = 0; i < k; i++) {

int a = rand() % (n -1) + 2;

double x = powermod( 2, d, n);

if( x ==1 || x == n-1)

continue;

for(int j = 1; j < s -1; j++) {

x = powermod( x, 2, n);

if( x == 1) return false;

if( x == n -1) continue;

return false;

}

}

return true; (probably)

}

**Growarray**

class Growarray {

int\* data;

int len;

int capacity;

void grow() { //O(len)

if(len < capacity) return;

int\* old = data;

data = new int[2\*capacity + 2];

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

data[i] = old[i];

capacity = 2\*capacity + 2;

delete [] old;

}

public:

Growarry() : data(nullptr), capacity(0), len(0) {}

~Growarry() { delete [] data; }

void addEnd(int v) { //O(1)

if(len >= capacity) grow();

data[len++] = v;

}

void addStart(int v) { //O(len)

if(len >= capacity) grow();

const int\* old = data;

data = new int[len + 1];

data[0] = v;

for(int i =0; i < len; ++i) {

data[i + 1] = old[i];

}

len++;

delete [] old;

}

void insert(inti , int v) { //O(len)

if(len >= capacity) grow();

const int\* old = data;

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

data[j] = old[j];

data[i] = v;

for(int j = i ; j < len; j++)

data[j + 1] = old[i];

len++;

delete [] old;

}

int removeEnd() { //O(1)

int e = data[len – 1];

len--;

return e;

}

void removeStart() { //O(len)

len--;

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

data[i] = data[i+1];

}

void remove(int i) { //O(len – i)

len--;

for(int j = i; j < len; j++)

data[j] = data[j + 1];

}

bool isEmpty() { return a.len == 0; )

};

**LinkedList 1**

class LinkedList1 {  
 struct Node{

int data;

Node\* next;

Node(int v, Node\* next) : data(v), next(next) {}

}

Node\* head;

public:

Linkedlist1 () : head(nullptr) {}

~LinkedList1() {

for(Node\* p = head; p != nullptr;) {

Node\* q = p;

delete [] q;

p = p->next;

}

}

void addStart(int v) { //O(1)

Node\* old = head;

head = new Node(v, old);

}

void addEnd(int v) { //O(n)

if(head == nullptr) {

head = new Node(v, nullptr); return;

}

Node\* p = head;

for(; p->next != nullptr; p = p->next) //move to the last

;

p->next = new Node(v, nullptr);

}

void removeStart() {

if(head == nullptr) return;

Node\* p = head;

head = head->next;

delete p;

return;

}

void removeEnd() { //O(n)

if(head == nullptr) return; //empty

Node\* p = head;

if(p->next == nullptr) { // only 1 element

head = nullptr;

delete p;

return;

}

//2+ elements

Node q = p->next;

for(; q ->next != nullptr; p = q; q = q->next) //p at last 2nd, q at last

;

delete q;

p->next = nullptr;

}

}

int start () {return head->data;}

bool isEmpty () { return head == nullptr; }

};

**LinkedList 2**

class LinkedList2 {  
 struct Node{

int data;

Node\* next;

Node(int v, Node\* next) : data(v), next(next) {}

}

Node\* head;

Node\* tail;

public:

Linkedlist2 () : head(nullptr) , tail(nullptr){}

~LinkedList2() {

for(Node\* p = head; p != nullptr;) {

Node\* q = p;

delete [] q;

p = p->next;

}

}

void addStart(int v) {

if(head == nullptr) {

head = new Node(v, nullptr);

tail = head;

}

else {

Node\* old = head;

head = new Node(v, old);

}

}

void addEnd(int v) {

if(tail == nullptr) {

head = new Node(v, nullptr);

tail = head;

}

else {

tail->next = new Node(v, nullptr);

tail = tail->next;

}

}

void removeStart() {

if(head == nullptr) return;

if(head == tail) {

delete head;

head = tail = nullptr;

return;

}

Node\* old = head;

head = head->next;

delete old;

}

void removeEnd() {

if(tail == nullptr) return;

if(head == tail) {

delete head;

head = tail = nullptr;

return;

}

Node\* p = head;

Node\* q = p->next;

for( ; q != tail; p = q; q= q->next) //q at last, p is q’s prev

;

tail = p;

tail->next = nullptr;

delete q;

}

};

**DoubleLinkedList 1**

class DoubleLinkedList1 {  
 struct Node{

int data;

Node\* next;

Node\* prev;

Node(Node\* prev, int v, Node\* next) : prev(prev), data(v), next(next) {}

}

Node\* head;

public:

DoubleLinkedlist1 () : head(nullptr) {}

~DoubleLinkedList1() {

for(Node\* p = head; p != nullptr;) {

Node\* q = p;

p = p->next;

delete [] q;

}

}

void addStart(int v) { //O(1)

Node\* oldhead = head;

Node\* t = new Node(nullptr, v, oldhead);

head = t;

head->prev = nullptr;

}

void addEnd(int v) { //O(n)

if(head == nullptr) {

Node\* t = new Node( nullptr, v, nullptr) ;

head = t;

return;

}

Node\* p = head;

for( ; p->next != nullptr; p = p->next)

;

Node\* t = new Node( p, v, nullptr);

p->next = t;

}

void removeStart() {

if(head == nullptr) return;

if(head->next == nullptr) {

Node\* p = head;

head = nullptr;

delete p;

return;

}

Node\* p = head;

head = head->next;

head->prev = nullptr;

delete p;

}

void removeEnd() { //O(n)

if(head == nullptr) return;

if(head->next == nullptr) {

Node\* p = head;

head = nullptr;

delete p;

return;

}

Node\* p = head;

for( ; p->next->next != nullptr; p = p->next)

;

Node\* last = p->next;

p->next = nullptr;

delete last;

}

};

**DoubleLinkedList 2**

class DoubleLinkedList2 {  
 struct Node{

int data;

Node\* next;

Node\* prev;

Node(Node\* prev, int v, Node\* next) :prev(prev), data(v), next(next) {}

}

Node\* head;

Node\* tail;

public:

DoubleLinkedlist2 () : head(nullptr) , tail(nullptr) {}

~DoubleLinkedList2() {

for(Node\* p = head; p != nullptr;) {

Node\* q = p;

p = p->next;

delete [] q;

}

tail = nullptr;

}

void addStart(int v) {

Node\* t = new Node(nullptr, v, head);

if(head == nullptr) {

head = tail = temp;

}

else {

head->prev = t;

head = t;

}

}

void addEnd(int v) {

Node\* t = new Node(tail, v, nullptr);

if(tail = nullptr)

tail = head = nullptr;

else {

tail->next = t;

tail = t;

}

}

void removeStart() {

if(head == nullptr) return;

if(head == tail) {

delete head;

head = tail = nullptr;

return;

}

Node\* t = head;

head = head->next;

head->prev = nullptr;

delete t;

}

void removeEnd() {

if(tail == nullptr) return;

if(tail == head) {

delete head;

head = tail = nullptr;

return;

}

Node\* seclast = tail->prev;

tail = seclast;

seclast->next = nullptr;

delete tail;

}

};

**Stack**

class Stack{

Growarry a;

public:

Stack() {}

void push(int v) { a.addEnd(v); }

void pop() { a.removeEnd(); }

int top() { return a.removeEnd(); }

bool isEmpty() { return a.isEmpry(); }

}

**Queue**

class Queue {

Linkedlist l;

public:

void enqueuer(int v) { l.addEnd(v); }

int dequeuer() { int n = l.start(); l.removeStart(); return n;}

bool isEmpty() { return l.isEmpty(); }

};