1. ***Programs for summation of series 1+X+X^2+X^3+…with different time***

***Complexities.***

## 1. **Naive Method (Loop) –**

We just calculate each power and add it to the sum.

#include <stdio.h>

#include <math.h>

int main() {

int n, x;

long long sum = 0;

printf("Enter value of x and n: ");

scanf("%d %d", &x, &n);

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

sum += pow(x, i); // computing each power

}

printf("Sum = %lld\n", sum);

return 0;

}

**Time Complexity:** (loop runs times).  
**Space Complexity:** .

## 2. **Exponentiation by Squaring (Fast Power) –**

Instead of computing each power normally, use **fast exponentiation** to speed up calculation.

#include <stdio.h>

// Fast power (x^n in O(log n))

long long power(long long x, int n) {

if (n == 0) return 1;

long long half = power(x, n/2);

if (n % 2 == 0) return half \* half;

else return x \* half \* half;

}

int main() {

int n, x;

long long sum = 0;

printf("Enter value of x and n: ");

scanf("%d %d", &x, &n);

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

sum += power(x, i); // faster power calculation

}

printf("Sum = %lld\n", sum);

return 0;

}

**Time Complexity:**

* Each power = , loop =
* Total = .  
  (Better for large exponents than pow()).

## 3. **Geometric Series Formula –**

The formula for sum of a geometric series is:

S = \frac{x^{n+1} - 1}{x - 1}, \quad \text{for } x \neq 1

S = n + 1

#include <stdio.h>

#include <math.h>

int main() {

int n, x;

long long sum;

printf("Enter value of x and n: ");

scanf("%d %d", &x, &n);

if (x == 1) {

sum = n + 1;

} else {

sum = (pow(x, n+1) - 1) / (x - 1);

}

printf("Sum = %lld\n", sum);

return 0;

}

**Time Complexity:** .  
**Space Complexity:** .

Naïve (Loop) → O(n)

Fast Power + Loop → O( n log n )

Formula → O(1)

2. Create a Binary Search Tree and perform the insertion, deletion operations.

### Binary Search Tree (BST)

* **Binary Search Tree property**:
  + Left child < Root
  + Right child > Root

### Operations:

1. **Insertion**
   * Compare value with root.
   * If smaller → go left.
   * If bigger → go right.
   * Insert at the correct leaf position.
2. **Deletion**
   * **Case 1:** Node has no child → simply delete it.
   * **Case 2:** Node has one child → replace node with its child.
   * **Case 3:** Node has two children → find the **inorder successor** (smallest value in right subtree), replace node’s value with it, and delete the inorder successor.

Code:

#include <iostream>

using namespace std;

struct Node {

int key;

Node \*left, \*right;

};

Node\* newNode(int item) {

Node\* temp = new Node;

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void inorder(Node\* root) {

if (root != NULL) {

inorder(root->left);

cout << root->key << " ";

inorder(root->right);

}

}

Node\* insert(Node\* node, int key) {

if (node == NULL) return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

return node;

}

Node\* minValueNode(Node\* node) {

Node\* current = node;

while (current && current->left != NULL)

current = current->left;

return current;

}

Node\* deleteNode(Node\* root, int key) {

if (root == NULL) return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else {

if (root->left == NULL) {

Node\* temp = root->right;

delete root;

return temp;

} else if (root->right == NULL) {

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

int main() {

Node\* root = NULL;

int values[] = {50, 30, 20, 40, 70, 60, 80};

for (int val : values)

root = insert(root, val);

cout << "Inorder traversal after insertion: ";

inorder(root);

cout << "\n\nDelete 20\n";

root = deleteNode(root, 20);

inorder(root);

cout << "\n\nDelete 30\n";

root = deleteNode(root, 30);

inorder(root);

cout << "\n\nDelete 50\n";

root = deleteNode(root, 50);

inorder(root);

return 0;

}

sample Output

Inorder traversal after insertion: 20 30 40 50 60 70 80

Delete 20

30 40 50 60 70 80

Delete 30

40 50 60 70 80

Delete 50

40 60 70 80