Name - Nikhil Kumar
College - NIT PATNA
Mob Number - 9801159223
Email Address - nikhil843113@gmail.com

(a)Compress 1:

First I have maintained the frequency of char c.
Whenever the c changes we insert the string in str.

Compress 2:

Obtained the char c with there frequency.
After that, I stacked the char with same frequency.

Decompress:

I have inserted the stacked into the result string according to there frequency.

Time complexity -
n = Len of string
Compress 1- O(n)
n = Len of compressed str
Compress 2- O(n)
n = Len of original str
Decompress - O(n)

Code -

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

using namespace std;

string compress1(string s){
      char c = s[0];
      int f = 1;
      string x;
      for(int i=1;i<s.length();i++){
         // cout<<i<<": "<<x<<"\n";
            if(c != s[i]){
                  x.push_back(c);
                  x += to_string(f);
                  c = s[i];
                  f = 1;
            }
            else
                  f++;
      }
      x.push_back(c);
      x += to_string(f);
      return x;
}
```

```
string compress2(string s){
      vector<char> v;
      vector<int> c;
      for(int i=0;i<s.length();){
            v.push_back(s[i]);
            i++;
            string x;
            while(isdigit(s[i])){
                  x.push_back(s[i]);
                  i++;
            }
            c.push_back(stoi(x));
      }

      string a;
      int f = c[0];
      a.push_back(v[0]);
      for(int i=1;i<v.size();i++){
            if(c[i] != f){
                  a += to_string(f);
                  a.push_back(v[i]);
                  f = c[i];
            }
            else
                  a.push_back(v[i]);
      }
      a += to_string(f);

      return a;
}

string obtain_original_string(string s){
      string x;
      for(int i=0;i<s.length();){
            string a, b;
            while(!isdigit(s[i])){
                  a.push_back(s[i]);
                  i++;
            }
            while(isdigit(s[i])){
                  b.push_back(s[i]);
                  i++;
            }
            int f = stoi(b);
            for(int j=0;j<a.size();j++){
                  for(int k=0;k<f;k++)
                        x.push_back(a[j]);
            }
      }
      return x;
}

int main(){
      string s;
      cin>>s;
      int l = s.length();

      string s1 = compress1(s);

      string s2 = compress2(s1);
      string s3 = obtain_original_string(s2);

      cout<<"Original string: "<<s<<"\n";
      cout<<"String aftre compression1: "<<s1<<"\n";
      cout<<"String after compression2: "<<s2<<"\n";
      cout<<"Decompressed string: "<<s3<<"\n";

    return 0;
}
```

(b) **time-complexcity-O(n),space-complexcity-O(1);**

This is the Optimized code

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


//This class is for creating a node inside memory
class node {
    public:
      int data;
      node* next;
      node(int val)
      {
            data = val;
            next = NULL;
      }
};

class llist {
    public:
      node* head;
      llist() { head = NULL; }

      //add node at the beginning
      void insertAtBegin(int val)
      {
            node* newNode = new node(val);
            newNode->next = head;
            head = newNode;
      }

// This fun will print nth node from end
      void nthFromEnd(int n)
      {
            node* main = head;
            node* ref = head;

            if (head == NULL) {
                  cout << "List is empty" << endl;
                  return;
            }

            for (int i = 1; i < n; i++) {
                  ref = ref->next;
                  if (ref == NULL) {
                        cout << n
                              << " is greater than no. of nodes in "
```

```
                                "the list"
                          << endl;
                    return;
              }
        }


        while (ref != NULL && ref->next != NULL) {
              ref = ref->next;
              main = main->next;
        }
        cout << "Node no. " << n
              << " from end is: " << main->data << endl;
  }
// To display all node
  void displaylist()
  {
        node* temp = head;
        while (temp != NULL) {
              cout << temp->data << "->";
              temp = temp->next;
        }
        cout << "NULL" << endl;
  }
};

int main()
{
  llist ll;

  ll.insertAtBegin(20);
  ll.insertAtBegin(4);
  ll.insertAtBegin(15);
  ll.insertAtBegin(35);

  ll.displaylist();

  ll.nthFromEnd(4);

  return 0;
}
```

(c) time complexcity-O(1) space-complexcity-O(n)
    This is the optimized code.

```
#include <bits/stdc++.h>
using namespace std;
class Solution{
    // This will store the min element
    int minEle;

    stack<int> s;
    public:

//this will return min element .
    int getMin(){

           if(s.empty())
           return -1;
```

```
            return minEle;
        }

// pop element from stack
        int pop(){

            int p=0;
            if(s.size()==0)
            return -1;
            int a=s.top();
            s.pop();

            if(a<minEle)
            {
                minEle=2*minEle-a;
              p= (a+minEle)/2;

            }
            else
            return a;

            return p ;
        }

        //add element in stack
        void push(int x){

            if(s.empty())
            {
            s.push(x);
            minEle=x;
            }
            else if(x<minEle)
            {
                s.push((2*x-minEle));
                minEle=x;
            }
            else
            s.push(x);
        }
};

int main()
 {
    long long t;

    cin>>t;
    while(t--)
    {
        int q;
        cin>>q;
        Solution ob;
        while(q--){
           int qt;
           cin>>qt;
           if(qt==1)
           {
              int att;
```

```
                cin>>att;
                ob.push(att);
            }
            else if(qt==2)
            {
                cout<<ob.pop()<<" ";
            }
            else if(qt==3)
            {
                cout<<ob.getMin()<<" ";
            }
        }
        cout<<endl;
    }
    return 0;
}
```

(d) **time-complexcity-O(n) space-complexcity-O(2*n)**

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

int trap(vector<int>& height) {
        int n=height.size();
        // To store max element present in the left to that index
        int left[n];

        // To store max element present in the right to that index
        int right[n];
        left[0]=height[0];
        for(int i=1;i<n;i++)
            left[i]=max(left[i-1],height[i]);

        right[n-1]=height[n-1];
        for(int i=n-2;i>=0;i--)
            right[i]=max(right[i+1],height[i]);

        int sum=0;
        for(int i=1;i<n-1;i++)
            sum+=min(left[i],right[i])-height[i];

        return sum;
    }

int main()
{
      vector<int> height={2,1,3,0,1,2,3};
      cout<<trap(height);
      return 0;
}
```

Question e)

To find the most optimum way to tender change using the least number of coins, you can use a greedy algorithm approach. This approach involves selecting the largest denomination coin possible until the desired change is reached.

Here's how you can implement this algorithm:

Sort the coin denominations in descending order.
Initialize an empty list to store the coins used for change.
Subtract the purchase amount from the payment amount to get the change amount.
While the change amount is greater than 0, do the following:
Iterate through the sorted coin denominations.
If the current denomination is less than or equal to the remaining change amount, subtract the denomination from the change amount and add the denomination to the list of coins used for change.
Return the list of coins used for change.
Let's say you have the following coin denominations: 25, 10, 5, and 1. If a customer purchases something for $37 and pays with a $50 bill, the change due is $13. Using the greedy algorithm approach, you would follow these steps:

Sort the coin denominations in descending order: [25, 10, 5, 1]
Initialize an empty list to store the coins used for change: []
Subtract the purchase amount from the payment amount to get the change amount: $50 - $37 = $13
While the change amount is greater than 0, do the following:
Iterate through the sorted coin denominations: 25, 10, 5, 1
If the current denomination is less than or equal to the remaining change amount, subtract the denomination from the change amount and add the denomination to the list of coins used for change.
$13 - 10 = $3, add 10 to the list of coins used for change: [10]
$3 - 1 = $2, add 1 to the list of coins used for change: [10, 1]
$2 - 1 = $1, add 1 to the list of coins used for change: [10, 1, 1]
$1 - 1 = $0, add 1 to the list of coins used for change: [10, 1, 1, 1]
Return the list of coins used for change: [10, 1, 1, 1]
In this scenario, the optimum way to tender the exact change to the customer using the least number of coins was to use one 10 cent coin, and three 1 cent coins.

Explain what is a greedy algorithm and how dynamic programming helps in this case?

A greedy algorithm is an approach to problem-solving that makes locally optimal choices at each step, with the hope of finding a global optimum solution. In other words, it makes the best possible decision at each step without considering the long-term consequences. Greedy algorithms are often used to solve optimization problems, such as finding the shortest path between two points or the most efficient way to allocate resources.

When it comes to coin problems, a greedy algorithm is often used to determine the minimum number of coins needed to make change for a given amount. The basic idea is to always use the largest coin denomination possible, until the remaining change can no longer be made with that denomination. Then, you move on to the next largest denomination and repeat the process until the desired change is reached. While this approach is not guaranteed to find the globally optimal solution, it often works well in practice and can be very efficient.

However, there are cases where a greedy algorithm may not work optimally. For example, suppose the coin denominations are 1, 5, and 10, and you need to make change for 15 cents. Using the greedy algorithm, you would use one 10-cent coin and one 5-cent coin, for a total of two coins. But the optimal solution is actually three 5-cent coins. In cases like this, a dynamic programming approach can be used to find the optimal solution.

Dynamic programming is a technique for solving complex problems by breaking them down into smaller, more manageable subproblems. It involves storing the solutions to subproblems in a table, so that they can be easily accessed and reused when needed. In the case of coin problems, dynamic programming can be used to find the minimum number of coins needed to make change for a given amount, by considering all possible combinations of coin denominations. This approach is guaranteed to find the globally optimal solution, but it can be slower and more memory-intensive than a greedy algorithm for small problem sizes. However, for larger problem sizes, dynamic programming can be much more efficient than a naive greedy approach.

Question - f)

Answer - The dot product and cross product are two fundamental operations used in vector mathematics, often used in computer graphics and other fields of engineering and physics.

The dot product, also known as the scalar product, is a mathematical operation that takes two vectors and produces a scalar quantity. It is defined as the product of the magnitudes of the two vectors and the cosine of the angle between them. In other words, it measures how much the two vectors point in the same direction. The dot product is used in a wide range of applications, such as computing the angle between two vectors, projecting one vector onto another, and determining whether two vectors are perpendicular.

The cross product, also known as the vector product, is a mathematical operation that takes two vectors and produces a third vector perpendicular to both of them. It is defined as the product of the magnitudes of the two vectors and the sine of the angle between them, where the direction of the resulting vector is determined by the right-hand rule. The cross product is used in computer graphics to compute the surface normal of a polygon, which is essential for shading and lighting calculations. It is also used in physics to calculate torque, angular momentum, and magnetic fields.

In summary, the dot product is used to determine how much two vectors point in the same direction, while the cross product is used to determine a vector that is perpendicular to both of them. In computer graphics, the dot product is used for lighting calculations and projection, while the cross product is used for calculating surface normals and determining the direction of rotations.

Reference Link - https://www.geeksforgeeks.org/dot-and-cross-products-on-vectors/

Question - g)

Answer - My favorite subject is Web development because:-
Creativity: Web development allows you to create visually appealing and interactive websites that can be accessed by anyone with an internet connection. This provides a platform for your creative expression and can be very satisfying.

Constantly evolving: The field of web development is constantly changing, with new technologies and frameworks emerging all the time. This means that there is always something new to learn and a new challenge to overcome.

Problem-solving: Web development requires a lot of problem-solving skills, as you need to figure out how to make the website work as intended, and how to fix bugs and errors. This can be very rewarding for people who enjoy logical thinking and puzzles.

Collaboration: Web development often involves working in teams, which can be a great way to build communication and collaboration skills. Working with others can also be a great way to learn new approaches and techniques.

Job prospects: There is high demand for skilled web developers, and this demand is expected to grow in the future. This means that there are plenty of job opportunities and career growth potential in this field.

Question - h)

Answer - There could be some general cases which may result in such behavior,

Random Variable : The application may be using some random variable, like random number generator or a specific time of the day or a user input etc which may be causing this.
Uninitialized Variable : May be there is some uninitialized variable, which takes an arbitrary value each time and those values are causing such drastic behavior.
Memory Leak : The program may have run out of memory, maybe heap overflow or something.
External Dependencies : The program may depend on some other application which is causing this.
Other process on machine : Maybe there are some other processes, running on machine causing it.
We can approach this problem by elimination, like close all other applications in the system or use some runtime tools to dig deeper when the problem occurs.

I declare that I have done the above work by myself and not worked with anyone or got help from any individual on the internet