Assignment 3

Answer to the question no.1

Let’s say we have an array of n elements starting from 1 to n. We have implement two stacks that do not overflow if the length of each stack is n so, we have to implement one stack from the end 1 to n and another one from n to 1. Stack uses LIFO policy.

For stack 1:

```
        Push(stack1,x){                                                   POP(stack1){
                If(top>n){print”overflow”}                                     top=top-1;
                top=top+1;                                                     return stack1[top+1]
}
                stack1[top]=x;
        }
```

For stack 2:

```
        Push(stack2 , x){
        If(top<1){print”overflow”}                //Overflow as the count is going from n to 1.
        top=top-1;
        stack2[top]=x;
}
POP(stack2){
        top=top+1;
        return stack[top-1];
}
```

Answer to the question no.2

S_1 and S_2 are two disjoint sets that has to be merged in O(1) time. We need the pointer to the last element of S_1 to do so. Otherwise, searching for last element is going to take O(n) time. We must implement the linked list as a stack with a pointer top pointing to the last element.

```
Last(S1){                                   //Just to get last element of S1
        Pointer=S1.head;
        While(pointer!="NULL"){
        Pointer=Pointer->next;
        }


}
UNION(S1,Pointer, S2){
        S=S1
        Pointer=S2;
}
```

Answer to the question no.3

Random variable X_{kl} represents the possible values of a collision occurrence of keys k !=l gets hashed to the same slot in the array T. The probability that a key gets hashed to a particular position is 1/m as there are m slots to be hashed into.

Therefore, The expectation value of both keys representing the same slot in the array is,

$E[X_{kl}]=1/m$

The collision can happen for combination of any two keys. We know that, (NC2) represents the number of distinct pairs of keys.

$\sum_{k\,!=l} E[X_{kl}]= (NC2) \text{ X } 1/m= N(N-1)/2 \text{ X } 1/m= N(N-1)/2m$

Answer to the question no.4

Hash table has to insert the values 5, 28, 19, 15, 20, 33, 12, 17, 10.

Using the Hash function,

H(k)=k mod 9

5: 5 mod 9=5 33: 33 mod 9=6

28: 28 mod 9=1 12: 12 mod 9=3

19: 19 mod 9=1 17: 17 mod 9=8

15: 15 mod 9=6 10: 10 mod 9=1

20: 20 mod 9=2

T is an array with position from 0 to 8. Array indexes 1, 2, 5, 6, 3 and 8 will only have the keys chained in the order they appeared in the question.

0:none

1: 10->19->28->none

2:20->none

3:12->none

4:none

5:5

6:33->15->none

7:none

8:17->none

Using the Hash function,

H(k)=Floor[m(k*A-Floor(K*A))]

m=1000

A=(√5-1)/2

KA mod 1 means the fractional part of KA-Floor[KA].

H(61)=Floor[1000*((61)* ((√5-1)/2)mod 1)]=Floor[1000*0.7000733]=700

H(62)=Floor[1000*((62)* ((√5-1)/2)mod 1)]=Floor[1000*0.318107302]=318

H(63)=Floor[1000*((63)* ((√5-1)/2)mod 1)]=Floor[1000*0.93614]=936

H(64)=Floor[1000*((64)* ((√5-1)/2)mod 1)]=Floor[1000*0.55417528]=554

H(65)=Floor[1000*((65)* ((√5-1)/2)mod 1)]=Floor[1000*0.172209]=172

T is an array with positions [1 to 100], among those only 5 position will be filled with those keys.

172:key 65

318:key 62

554:key 64

700:key 61

936:key 63

All the other array positions will be empty.

Answer to the question no.5

Part A:

2-Universal hash functions contain two keys so for the keys X!=Y in a hash function {X,Y} can be of any sequence in the set $\{0,1,\ldots\ldots,m\}^2$. As the hash function h changes within the set of hash functions H, the count of collisions (h(X)=h(Y)) is (1/m) |H|, and H is universal.

Therefore, if a ser=t H is 2-universal then this class H is universal class too.

Part B:

In the hash class for a set of keys {m,m'} all the hash value sets{h(m),h(m')} are selected with equal likelihood when the hash function h is selected at random. H will be 2-universal so all pairs p{t,h(m')} in the hash class are equally probable. If the adversary knows hash class H then seeing the key will tell him about the function. All p cases have the same probability, and which equals 1. The probability is 1/p, and the adversary can do nothing better than guessing.

Answer to the question no.6

Pseudo-code:

```
preorder(root){
if(root==){return;}
preorder(root->left);
preorder(root->right);
}
```

Correctness:

Any node's string is a prefix of all its descendant strings and hence belongs before them in the sorted order. Node's left descendants belong before it's right descendants because the corresponding strings are identical till parent nodes and left subtree's strings have 1.

Time:

Insertion takes BIGTHETA(n) time as insertion of each string takes time proportional to it's length and sum of all string lengths is n. The preorder tree walk takes O(n) time. All the strings have length n and length l string corresponds to a path through the root and l other nodes but a single node may be shared among many string paths.

Answer to the question no.7

```
Memoized_Cutrod(p,k){
        int A[k];
        int result[k];
        for(int i=1;i<=k;i++){
        A[k]=-100000;
        Auxiliary_function(p,k,A, result);
        for(int i=0;i<=n;i++){
                print(result [n]);
        }
        }
}
Auxiliary_function(p,k,A,B){
        if(A[k]>=0){
                return A[k];
        }
        if(k==0){
                w=0;
        }else{
                W=-1000;
        }
        for(int i=0;i<=k;i++){
                if(w<p[v]+Auxiliary_function(p,k-I,A, result)){
                        w= p[v]+Auxiliary_function(p,k-I,A, result);
                }
        }
        result [k]=i;
        A[k]=w;
        return w, result;
}
```

Answer to the question no.8

Cut-Rod:

```
#include<iostream>
#include<time.h>
using namespace std;
int cutrod(int price[],int size);
int main(){
  int size;
  cout<<"Please enter the size: ";
  cin>>size;
  int* price=new int[size];
  for(int i=0;i<=size;i++){
    price[i]=i;

    }
    //int size = sizeof(arr)/sizeof(arr[0]);
    clock_t start=clock();
  int result=cutrod(price,size);
  clock_t end=clock();
  cout<<"The result is :"<<result;
  cout<<"Time taken"<<(double)(end - start)/CLOCKS_PER_SEC;
  return 0;
}

int cutrod(int price[],int size){

if (size <= 0)
     return 0;
   int max_val = INT_MIN;


   for (int i = 0; i<size; i++)
         max_val = max(max_val, price[i] + cutrod(price, size-i-1));

   return max_val;
}

int max(int a, int b) { return (a > b)? a : b;}
```

```
For size: 5 10 15 20 25 30

Runtimes:0.001 0.002 0.003 0.009 0.19 0.28
```

Memoized cut-rod:

```
#include<iostream>
#include<time.h>
using namespace std;

int memoizedcutrod(int price[],int size);
int auxiliary(int price[],int size,int revenue[]);
int main(){
  int size;
  cout<<"Please enter the size: ";
  cin>>size;
  int* price=new int[size];
  for(int i=0;i<=size;i++){
    price[i]=i+1;
    cout<<price[i];

    }
    //int size = sizeof(arr)/sizeof(arr[0]);
    clock_t start=clock();
  int result=memoizedcutrod(price,size);
  clock_t end=clock();
  cout<<"The result is :"<<result;
  cout<<"Time taken"<<(double)(end - start)/CLOCKS_PER_SEC;
  return 0;
}
int max(int a, int b) { return (a > b)? a : b;}
int memoizedcutrod(int price[],int size){

    int* revenue=new int [size+1];
    for(int i=0;i<size+1;i++){

        revenue[i]=INT_MIN;
    }
    return auxiliary(price,size,revenue);
}

int auxiliary(int price[],int size,int revenue[]){
    int q;
    if(revenue[size]>=0){
        cout<<"HUUUR";
        return revenue[size];
    }
    if(size=0){
        q=0;
    }else{
        q=INT_MIN;

    for(int i=1;i<size;i++){
        q=max(q,price[i]+auxiliary(price,size-i-1,revenue));
    }
    }
    revenue[size]=q;
    cout<<q;
    return q;
}
```

For size: 5 10 15 20 25 30

Runtimes: 0.0 0.0 0.001 0.003 0.010 0.10

Bottom_up cut-rod:

```
#include<iostream>
#include<time.h>
using namespace std;
int BOTTOM_UP_CUT_ROD(int price[],int size);
int main(){
  int size;
  cout<<"Please enter the size: ";
  cin>>size;
  int* price=new int[size];
  for(int i=0;i<=size;i++){
    price[i]=i;

    }
    //int size = sizeof(arr)/sizeof(arr[0]);
    clock_t start=clock();
  int result=BOTTOM_UP_CUT_ROD(price,size);
  clock_t end=clock();
  cout<<"The result is :"<<result;
  cout<<"Time taken"<<(double)(end - start)/CLOCKS_PER_SEC;
  return 0;
}

int BOTTOM_UP_CUT_ROD(int price[],int size){
    int r[size+1];
    int q;
    r[0] = 0;
    for (int j = 1; j <= size; j++)
    {
        q = INT_MIN;
        for (int i = 0; i < j; i++)
        {
            q = max(q,price[i]+r[j-i-1]);
        }
        r[j] = q;
    }
    return r[size];
}

int max(int a, int b) { return (a > b)? a : b;}
```

```
For size: 5 10 15 20 25 30

Run Times: 0.0 0.0 0.001 0.001 0.011 0.012
```