CMPT307 ASSIGNMENT 3 (RITIKA GOYAL) (301401516)

Ques1:

The two stacks are implemented on the same array A[0 …… n-1]. The first stack, satck1, is pointed at -1 initially when it is empty and then it grows with the increasing index of array. The second stack, stack2, initially points at index n when it is empty and grows with the decreasing index of array. When the top of both the stacks are consecutive, it means all the elements of array are in stack which prevents overflow by eliminating any push. When the stacks are empty, it eliminates pop function.

Let St be the stack we want to push or pop from and element be the parameter that needs to be inserted or deleted.

<u>Pseudo Code:</u>

```
Input: Stack to be pushed, element to be pushed in a stack
Output: insert the element at the top of stack
 push( St, element)
        if St == stack1 then
                if  stack1.top + 1  != stack2.top then
                        stack1.top = stack1.top+1  and  stack1[stack1.top] = element;
                else
                        error “overflow”

        else if St == stack2 then
                if  stack2.top -1 != stack1.top  then
                        stack2.top = stack2.top -1;
                        stack2[stacl2.top] = element;
                else
                        error “overflow”


Input: stack to be popped
Output: returns the element that is taken out of stack and reduces the top of stack
pop(St)

if St == stack1
        if stack1.top == -1 then error “underflow”
        else
                temp = stack1[stack1.top]
                stack1.top = stack1.top-1;
                return temp;

else if St == stack2
        if stack2.top == n then “underflow”
        else
                temp = stack2[stack2.top];
                stack2.top = stack2.top+1;
                return temp;
```

Ques2:

Let S1 and S2 be the two sets that are disjoint and S be the set such that S = S1 U S2. We consider S1 and S2 to be the doubly linked list and check if either of them is null and if not then point S1.tail.next to S2.head and S2.head.prev = S1.tail and then simply point S.head = S1.head.

Pseudo-code

```
union (S, S1, S2)
    if S1 != null && S2 != null   then
        S.head = S1.head;
        S1.tail.next = S2.head;
        S2.head.prev = S1.tail;

    else if S1 == null
        S.head = S2.head

    else if S2 == null
        S.head = S1.head
    return S
```

Running Time: T(n) = T(1) = O(1)

Ques3:

Total elements in array = m

Total keys to be hashed using has function = n

Let X be the expected value of collision occurrence such that for keys $k \neq l$, $h(k) = h(l)$

Due to simple uniform hashing,

Probability[h(k) = h(l)] = 1/m.

=> The expected value of two keys to collide = 1/m.

Total expected collisions are = $\sum_{k\neq l} E[X]$

$= \sum_{k\neq l} 1/m$

$= \binom{n}{2}$ 1/m

$= \frac{n(n-1)}{2m}$

Ques4:

a) Indexes ->

0	1	2	3	4	5	6	7	8
	28	20	12		5	15		17
	19					33		
	10							

b)

key	61	62	63	64	65
Mapped index	700	318	936	554	172

Ques5:

a) H is 2-universal if for every fixed sequence of distinct key (x_1, x_2) and for any hash function h chosen at random form H, the sequence ($h(x_1)$, $h(x_2)$) is equally likely to be any of m^2 sequences of length 2 with elements drawn from {0, 1, …, m-1}.
Total number of sequences = m^2
The possible values of x (where x belongs to [m]) which results in collision = m
Probability [collision occurs that is $h(x_1) = h(x_2)$] = $\frac{m}{m^2}$
$= \frac{1}{m}$ <= 1 (satisfying the condition of Universal hash function)
Therefore, if H is 2-universal then H is universal.

b) Since H is 2-universal, any key pair (k, k') are equally likely to map at (h(k), h(k')) when h is chosen at random.
Since Z_p = {0, 1, …, p-1}, So the total p cases are equally likely, and their sum is 1.
Since we have two distinct keys k and k', Therefore:
Probability [fooling Bob] = P [choosing t' and k']
$= \frac{1}{p} * {}^2C_1$
$= \frac{1}{p} * 2$
$= \frac{2}{p}$

Ques6:

The elements are inserted in radix tree and then pre-order traversal is used to sort the elements in lexicographical order.

<u>Pseudo-Code:</u>

```
/* used pre-order traversal to print in sorted values*/
PrintInSortedOrder(Node* root)
        if root = Null then return;
        if root->str != "" then cout<<root->str<<endl;
        PrintInSortedOrder(root->left);
        PrintInSortedOrder(root->right);
```

<u>Correctness:</u>

Loop Invariant - At start of each recursive iteration, the root is null or do not have any value.
Initialisation: Pre order is correct for empty tree as it checks if the root of the tree is empty and if yes then it returns null.
Maintenance: When the tree is not empty, then first the string of the root is printed and then it first traverse to left and then traverse to right recursively printing the data of the nodes. Since the algorithm is recursive, every time it gets the new node, it checks if it is null or the data in it is null and hence the invariant is true.
Termination: The function is terminated when all the values of data in the tree is printed and returns NIL.

<u>Running Time:</u>

T(n) = 2-recursive call to traverse left and right sub tree + base case
T(n) = 2* T(n/2) + T(c)
$= 2 * T(\frac{n}{2})$ + T(1) = O(n).

Ques7:

```
Memoized-Cut-Rod(p,n)
    let r[0 …. n] be a new array which stores value;
    let s[0 …. n] be a new array which return actual solution initialized ;
    for i = 0 to n do r[i] = -∞;
    return Memoized-Cut-Rod-Aux(p, n, r, s);

Memoized-Cut-Rod-Aux(p, n, r, s)
    if r[n] ≥ 0 then return r[n];
    if n = 0 then q = 0 else q = -∞;
    for i = 1 to n do
        if q < p[i] + Memoized-Cut-Rod-Aux(p, n-i, r, s) then
            q = p[i] + Memoized-Cut-Rod-Aux(p, n-i, r, s) and s[n] = i;
    r[n] = q;
    return q and s;
```

Ques8:

	Cut-Rod	**Memoized-Cut-Rod**	**Bottom-Up-Cut-Rod**
N = 5	0.000002 seconds	0.000001 seconds	0.000001 seconds
N = 10	0.00001 seconds	0.000001 seconds	0.000001 seconds
N = 15	0.000265 seconds	0.000002 seconds	0.000001 seconds
N = 20	0.006816 seconds	0.000004 seconds	0.000002 seconds
N = 25	0.240878 seconds	0.000004 seconds	0.000002 seconds
N = 30	8.59796 seconds	0.000006 seconds	0.000003 seconds

The comparison based on execution time:

Bottom-Up-Cut-Rod < Memoized-Cut-Rod < Cut-Rod

Source-Code:

```
#include <iostream>
#include <ctime>
#include <limits>
using namespace std;


int Cut_Rod(int* p, int n)
{
    if (n == 0)
        return 0;
    int q = std::numeric_limits<int>::min();
    for(int i =1 ; i<n+1 ;i++)
    {
        int temp = p[i] + Cut_Rod(p,n-i);
        if(q< temp)
```

```
                        q = temp;
        }
        return q;
}

int Memoized_Cut_Rod_Aux(int *p, int n, int* r)
{
        int q;
        if(r[n]>=0)
                return r[n];
        if(n==0)
                q = 0;
        else
        {
                q = std::numeric_limits<int>::min();
        }

        for(int i =1 ; i<n+1 ; i++)
        {
                int temp = p[i] + Memoized_Cut_Rod_Aux(p,n-i,r);
                if(q<temp)
                        q = temp;
        }

        r[n] = q ;
        return q;
}
int Memoized_Cut_Rod(int* p ,const int n)
{
        int r [n+ 1];
        for(int i =0 ; i<n+1;i++)
        {
                r[i] = std::numeric_limits<int>::min();
        }


        return Memoized_Cut_Rod_Aux(p,n,r);
}


int Bottom_Up_Cut_Rod(int * p, const int n)
{
        int r[n+1];
        r[0] = 0;
        int q;
        for(int j = 1 ; j <= n;j++)
        {
                q = std::numeric_limits<int>::min();
                for (int i = 1; i <=j; ++i)
                {
                        if(q < p[i] + r[j-i])
                                q = p[i] + r[j-i];
                }
                r[j] = q;
        }


        return r[n];

}

int main()
{
        int p[] = {0,1,5,8,9,10,17,17,20,24,30,32,40,45,67,68,68,70,72,73,75,78};

        for(int i =1 ; i<=30 ; i++)
        {
                if(i%5 == 0)
                {
                        cout<<"For N = "<<i<<" :"<<endl;
                        clock_t cutRod = clock();
                        int sol1 = Cut_Rod(p,i);
                        cutRod = clock() - cutRod;
                        cout<<"Time taken by Cut-Rod = "<<(float)cutRod/ CLOCKS_PER_SEC<<"
seconds"<<endl;

                        clock_t MemoizedCutRod = clock();
                        int sol2 = Memoized_Cut_Rod(p,i);
                        MemoizedCutRod = clock() - MemoizedCutRod;
                        cout<<"Time taken by Memoized-Cut-Rod = "<<(float)MemoizedCutRod/
CLOCKS_PER_SEC<<" seconds"<<endl;

                        clock_t BottomUpCutRod = clock();
                        int sol3 = Bottom_Up_Cut_Rod(p,i);
                        BottomUpCutRod = clock() - BottomUpCutRod;
                        cout<<"Time taken by Bottom-Up-Cut-Rod = "<<(float)BottomUpCutRod/
CLOCKS_PER_SEC<<" seconds"<<endl<<endl;
                }
        }
        return 0;
}
```