Week 6 – Assignments

1. Explain in detail and Write a C program for Belady's anomaly. (A/P)

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

using namespace std;

void pageFault(int frame_size, int* ref, int len)
{
	// To dynamically allocate an array,
	// it represents the page frames.
	int* arr = new int[frame_size];

	// To initialize the array
	for (int i = 0; i < frame_size; i++) {
		arr[i] = -1;
	}

	// To count page faults
	int cnt = 0;
	int start = 0;
	int flag;
	int elm;

	for (int i = 0; i < len; i++) {
		elm = ref[i];
```

```
            // Linear search to find if the page exists
            flag = 0;
            for (int j = 0; j < frame_size; j++) {
                if (elm == arr[j]) {
                    flag = 1;
                    break;
                }
            }
            // If the page doesn't exist it is inserted,
            // count is incremented
            if (flag == 0) {
                if (start < frame_size) {
                    arr[start] = elm;
                    start++;
                }
                else if (start == frame_size) {
                    arr[0] = elm;
                    start = 1;
                }
                cnt++;
            }
        }
        cout << "When the number of frames are: " << frame_size << ", ";
        cout << "the number of page faults is : " << cnt << endl;
}
```

```
int main()
{
	// Reference array
	int ref[] = { 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 };
	int len = sizeof(ref) / sizeof(ref[0]);
	// The frame size
	int frame_size = 3;

	pageFault(frame_size, ref, len);

	// Increase value of frame size
	frame_size = 4;

	// The page fault increases
	// even after increasing the
	// the number of frames.
	// This is Belady's Anomaly
	pageFault(frame_size, ref, len);
}
```

2. Explain in detail and Write a C program for Lazy Buddy System Algorithm (A/P)

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

// Size of vector of pairs
```

```
int size;

// Global vector of pairs to store
// address ranges available in free list
vector<pair<int, int>> free_list[100000];

// Map used as hash map to store the starting
// address as key and size of allocated segment
// key as value
map<int, int> mp;

void initialize(int sz)
{

    // Maximum number of powers of 2 possible
    int n = ceil(log(sz) / log(2));
    size = n + 1;

    for(int i = 0; i <= n; i++)
        free_list[i].clear();

    // Initially whole block of specified
    // size is available
    free_list[n].push_back(make_pair(0, sz - 1));
}
```

```
void allocate(int sz)
{

	// Calculate index in free list
	// to search for block if available
	int n = ceil(log(sz) / log(2));

	// Block available
	if (free_list[n].size() > 0)
	{
		pair<int, int> temp = free_list[n][0];

		// Remove block from free list
		free_list[n].erase(free_list[n].begin());
		cout << "Memory from " << temp.first
			<< " to " << temp.second << " allocated"
			<< "\n";

		// map starting address with
		// size to make deallocating easy
		mp[temp.first] = temp.second -
						temp.first + 1;
	}
	else
	{
		int i;
```

```
for(i = n + 1; i < size; i++)
{

        // Find block size greater than request
        if(free_list[i].size() != 0)
                break;
}

// If no such block is found
// i.e., no memory block available
if (i == size)
{
        cout << "Sorry, failed to allocate memory \n";
}

// If found
else
{
        pair<int, int> temp;
        temp = free_list[i][0];

        // Remove first block to split it into halves
        free_list[i].erase(free_list[i].begin());
        i--;

        for(; i >= n; i--)
```

```
{

        // Divide block into two halves
        pair<int, int> pair1, pair2;
        pair1 = make_pair(temp.first,
                                        temp.first +
                                        (temp.second -
                                        temp.first) / 2);
        pair2 = make_pair(temp.first +
                                        (temp.second -
                                        temp.first + 1) / 2,
                                        temp.second);

        free_list[i].push_back(pair1);

        // Push them in free list
        free_list[i].push_back(pair2);
        temp = free_list[i][0];

        // Remove first free block to
        // further split
        free_list[i].erase(free_list[i].begin());
}
cout << "Memory from " << temp.first
        << " to " << temp.second
        << " allocated" << "\n";
```

```
                    mp[temp.first] = temp.second -
                                                    temp.first + 1;
            }
        }
}

// Driver code
int main()
{

    // Uncomment following code for interactive IO
    /*
    int total,c,req;
    cin>>total;
    initialize(total);
    while(true)
    {
            cin>>req;
            if(req < 0)
                    break;
            allocate(req);
    }*/

    initialize(128);
    allocate(32);
```

```
        allocate(7);
        allocate(64);
        allocate(56);

        return 0;
    }

    // This code is contributed by sarthak_eddy
```

3. Explain in detail and Write a C program for Additional Reference Bits Algorithm

```
// C code to implement the approach
#include <stdio.h>

// Function to reverse bits of num
unsigned int reverseBits(unsigned int num)
{
    unsigned int NO_OF_BITS = sizeof(num) * 8;
    unsigned int reverse_num = 0;
    int i;
    for (i = 0; i < NO_OF_BITS; i++) {
        if ((num & (1 << i)))
            reverse_num |= 1 << ((NO_OF_BITS - 1) - i);
    }
    return reverse_num;
}
```

```
// Driver code
int main()
{
        unsigned int x = 2;
        printf("%u", reverseBits(x));
        getchar();
}
```

4. Explain in detail and Write a C program for Second Chance Algorithm (A/P)

```
// CPP program to find largest in an array
// without conditional/bitwise/ternary/ operators
// and without library functions.
#include<iostream>
#include<cstring>
#include<sstream>
using namespace std;

// If page found, updates the second chance bit to true
static bool findAndUpdate(int x,int arr[],
                                bool second_chance[],int frames)

{
    int i;

    for(i = 0; i < frames; i++)
```

```
        {

                if(arr[i] == x)
                {
                        // Mark that the page deserves a second chance
                        second_chance[i] = true;

                        // Return 'true', that is there was a hit
                        // and so there's no need to replace any page
                        return true;
                }
        }

        // Return 'false' so that a page for replacement is selected
        // as he reuested page doesn't exist in memory
        return false;

}


// Updates the page in memory and returns the pointer
static int replaceAndUpdate(int x,int arr[],
                        bool second_chance[],int frames,int pointer)
{
        while(true)
        {
```

```
                // We found the page to replace
                if(!second_chance[pointer])
                {
                        // Replace with new page
                        arr[pointer] = x;

                        // Return updated pointer
                        return (pointer + 1) % frames;
                }

                // Mark it 'false' as it got one chance
                // and will be replaced next time unless accessed again
                second_chance[pointer] = false;

                //Pointer is updated in round robin manner
                pointer = (pointer + 1) % frames;
        }
}

static void printHitsAndFaults(string reference_string,
                                                                int
frames)
{
        int pointer, i, l=0, x, pf;
```

```
//initially we consider frame 0 is to be replaced
pointer = 0;

//number of page faults
pf = 0;

// Create a array to hold page numbers
int arr[frames];

// No pages initially in frame,
// which is indicated by -1
memset(arr, -1, sizeof(arr));

// Create second chance array.
// Can also be a byte array for optimizing memory
bool second_chance[frames];

// Split the string into tokens,
// that is page numbers, based on space

string str[100];
string word = "";
for (auto x : reference_string)
{
        if (x == ' ')
        {
```

```
                str[l]=word;
                word = "";
                l++;
        }
        else
        {
                word = word + x;
        }
}
str[l] = word;
l++;
// l=the length of array

for(i = 0; i < l; i++)
{
        x = stoi(str[i]);

        // Finds if there exists a need to replace
        // any page at all
        if(!findAndUpdate(x,arr,second_chance,frames))
        {
                // Selects and updates a victim page
                pointer = replaceAndUpdate(x,arr,
                                second_chance,frames,pointer);

                // Update page faults
```

```
                        pf++;
                }
        }
        cout << "Total page faults were " << pf << "\n";
}

// Driver code
int main()
{
        string reference_string = "";
        int frames = 0;

        // Test 1:
        reference_string = "0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4";
        frames = 3;

        // Output is 9
        printHitsAndFaults(reference_string,frames);

        // Test 2:
        reference_string = "2 5 10 1 2 2 6 9 1 2 10 2 6 1 2 1 6 9 5 1";
        frames = 4;

        // Output is 11
        printHitsAndFaults(reference_string,frames);
        return 0;
```

```
}


// This code is contributed by NikhilRathor
```