BÀI TẬP RECURSION

Bài 1:

a)

```
int f(int n)

{

    if (n == 1)

        return 1;

    return f(n-1)*2;

}
```

- Bước 1: Base case là gì ?

```
if (n == 1)

        return 1;
```

- Bước 2: Kết quả base case ?

Khi n = 1 hàm trả về 1.

- Bước 3: Trước Base case ?

Khi n = 2 hàm trả về f(2) = 2.

- Bước 4: Trước của trước base case ?

Khi n = 3 hàm trả về f(3) = 4

b)

```
float g(int n)

{

    if (n == 1)
```

```
        return 1.0;

    return n * n + g(n - 1);

}
```

- Bước 1: Base case là gì ?

```
if (n == 1)

        return 1.0;
```

- Bước 2: Kết quả base case ?

Khi n = 1 hàm trả về 1.0

- Bước 3: Trước Base case ?

Khi n = 2 hàm trả về g(2) = 5.0

- Bước 4: Trước của trước base case ?

Khi n = 3 hàm trả về g(3) = 14.0

c)

```
int F(int n)

{

    if (n == 1) return 1;

if (n == 2) return 1;

    return F(n-1)+F(n-2);

}
```

- Bước 1: Base case là gì ?

```
if (n == 1) return 1;
if (n == 2) return 1;
```

- Bước 2: Kết quả base case ?

Khi n = 1 hàm trả về 1.
Khi n = 2 hàm trả về 1;

- Bước 3: Trước Base case ?

Khi n = 2 hàm trả về F(2) = 1.

- Bước 4: Trước của trước base case ?

Khi n = 3 hàm trả về F(3) = 2

d)

- Bước 1: Base case là gì ?

```
    if (n == 0) return 1;
    if (n == 2) return 2;
```

- Bước 2: Kết quả base case ?

Khi n = 0 hàm trả về 1.
Khi n =2 hàm trả về 2.

- Bước 3: Trước Base case ?

Khi n = 2 hàm trả về P(3) = 2.

- Bước 4: Trước của trước base case ?

Khi n = 3 hàm trả về P(4) = 3.

Bài 2:

1: Viết hàm đệ quy in một mảng số nguyên a có n phần tử theo thứ tự từ a[0] đến a[n-1] (đầu đến cuối).

```
void recursion(int a[], int k, int n) {

    if(k == n) {

        return;

    }

    printf("%d ", a[k]);

    recursion(a, k + 1, n);
```

```
}
```

với k là biến chạy, n là số phần tử trong mảng.

2: Dựa trên thuật toán Linear Search, hãy cài đặt một hàm đệ quy để tìm một phần tử trong mảng số nguyên a và xuất ra chỉ số của phần tử đó.

```
int recursion(int a[], int k, int n, int value) {

    if(k == n) {

        return -1;

    }

    if(a[k] == value) {

        return k;

    }

    return recursion(a, k + 1, n, value);

}
```

với k là biến chạy, n là số phần tử trong mạng và value là giá trị cần tìm. Nếu mảng không có giá trị cần tìm thì trả về -1.

3

a) Cho một stack đầy có 20 phần tử. Hãy viết hàm đệ quy để pop lần lượt các phần tử từ stack cho đến khi stack rỗng.

```
#include<stdio.h>



#define MAX 30



typedef struct

{
```

```
    int a[MAX];
    int top;
} Stack;


void init(Stack *s){
    s->top = -1;
}


int isEmpty(Stack* s) {
    if(s->top == -1) {
        return 1;
    }else return 0;
}
int isFull(Stack* s) {
    if(s->top == MAX - 1) {
        return 1;
    }
    return 0;
}
void push(Stack* s, int value) {
    s->a[++s->top] = value;
}
int pop(Stack* s) {
    int value = s->a[s->top];
```

```
    --s->top;
    return value;
}
void displayStack(Stack* s) {
    printf("\nStack: ");
    for(int i = 0; i <= s->top; i++) {
        printf("%3d ", s->a[i]);
    }
    printf("\n");
}

void popStack(Stack *s) {
    if(isEmpty(s)) {
        return;
    }else {
        pop(s);
        popStack(s);
    }
}

int main() {
    Stack s;
    init(&s);
    for(int i = 0; i < 20; i++) {
        push(&s, i);
```

```
    }

    popStack(&s);

    printf("%d\n", isEmpty(&s));
    // Kiem tra xem ham pop co hoat dong tot hay khong

    return 0;

}
```

Hàm cần làm là hàm `popStack`.

b)Áp dụng tương tự với queue.

```
#include <stdio.h>

#include<stdlib.h>

#define MAX 21

#define null INT_MIN

typedef struct {

    int head, tail;

    int a[MAX];

} Queue;



void init(Queue* q) {

    q->head = 0;

    q->tail = -1;

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

        q->a[i] = null;

    }

}
```

```
int isEmpty(Queue *q) {
    if(q->head == 0 && q->tail == -1) {
        return 1;
    }
    return 0;

}

void put(Queue* q, int value) {
    if(q->tail < MAX - 1) { // 5
        q->tail += 1;
        q->a[q->tail] = value;
    }else {
        q->tail = -1;
        if(q->tail + 1 < q->head) {
            q->tail += 1;
            q->a[q->tail] = value;
        }else {
            q->tail = MAX - 1;
            // printf("Queue is full!\n");
        }
    }
}
```

```
int get(Queue* q) {
    if(isEmpty(q)) {
        return null;
    }
    int index = q->head;
    int value;
    if(q->head == q->tail) {
        // bao dong
        q->tail = -1;
        q->head = 0;
        value = q->a[index];
        q->a[index] = null;
    }else if(index == MAX - 1){
        value = q->a[index];
        q->a[index] = null; // danh dau phan tu da bi xoa
        q->head = 0;
    }else {
        value = q->a[index];
        q->a[index] = null;
        q->head += 1;
    }

    return value;
}
```

```
void popQueue(Queue *q) {
        if(isEmpty(q)) {
                return;
        }else {
                get(q);
                popQueue(q);
        }
}

int main() {

    Queue q;

    init(&q);

    for(int i = 0; i < 20; i++) {
            put(&q, i);
    }

        popQueue(&q);

        printf("%d", isEmpty(&q));
        // kiem tra xem ham co hoat dong tot khong
    return 0;



}
```

Hàm cần làm là hàm `popQueue`

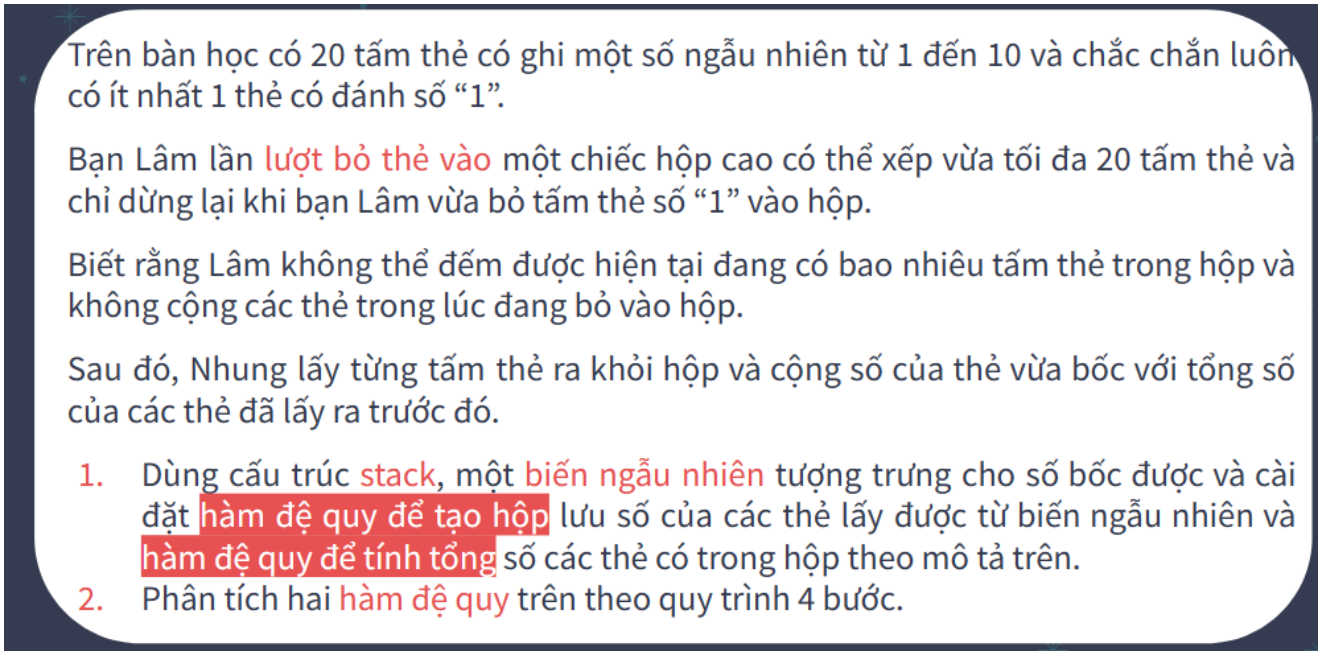
4: Dựa trên thuật toán Binary Search, hãy cài đặt một hàm đệ quy để tìm một phần tử trong mảng số nguyên a và xuất ra chỉ số của phần tử đó.

```
#include<stdio.h>



int binarySearch(int a[100], int value, int left, int right) {

    if(left <= right) {

        int mid = (left + right) / 2;

        if(a[mid] == value) return mid;
```

```
        else if(a[mid] > value) {
            return binarySearch(a, value, left, mid - 1);
        }else return binarySearch(a, value, mid + 1, right);
    }else {
        return -1;
    };
}

int main() {
    int a[] = {1, 2, 3, 4, 5};
        printf("%d   %d",binarySearch(a, 2, 0, 4), binarySearch(a, 6, 0, 4))
}
```

```
output : 1  -1
```

Bài 3:




```
#include<stdio.h>
```

```c
#define MAX 30


typedef struct
{
    int a[MAX];
    int top;
} Stack;


void init(Stack *s){
    s->top = -1;
}


int isEmpty(Stack* s) {
    if(s->top == -1) {
        return 1;
    }else return 0;
}
int isFull(Stack* s) {
    if(s->top == MAX - 1) {
        return 1;
    }
    return 0;
}
void push(Stack* s, int value) {
```

```
    s->a[++s->top] = value;
}
int pop(Stack* s) {
    int value = s->a[s->top];
    --s->top;
    return value;
}
void displayStack(Stack* s) {
    printf("\nStack: ");
    for(int i = 0; i <= s->top; i++) {
        printf("%3d ", s->a[i]);
    }
    printf("\n");
}


//---------------------------------------
Stack s;


void recursion1(int *a) {
    if(*a == 1) {
        push(&s, 1);
        printf("STOPED\n");
        return;
    }
    push(&s, *a);
```

```
    recursion1(++a);
}


int recursion2(Stack *s) {
    if(isEmpty(s)) {
        return 0;
    }
    return pop(s) + recursion2(s);
}


int main() {

    init(&s);
    int a[20] = {10, 9, 8, 5, 6, 2, 3, 1, 6, 7, 1, 5, 6, 7, 5, 4, 2, 1, 10, 8};
    recursion1(a);
    int sum = 0;
    for(int i = 0; i < 8; i++) {
        sum += a[i];
    }
    printf("Sum use for loop is %d\n", sum);
    printf("Sum use recursion is %d\n", recursion2(&s));
    return 0;
}
```

2 Hàm đề yêu cầu là `recursion1` và `recursion2`