DSA Tutorial 3 – Section A
IIITD

Q1. Suppose you are provided with the following function declaration in the C programming language.

int partition(int a[], int n);

The function treats the first element of a[ ] as a pivot and rearranges the array so that all elements less than or equal to the pivot are in the left part of the array, and all elements greater than the pivot are in the right part. In addition, it moves the pivot so that the pivot is the last element of the left part. The return value is the number of elements in the left part. The following partially given function in the C programming language is used to find the k^{th} smallest element in an array a[ ] of size n using the partition function. We assume $k \leq n$.

```
int kth_smallest (int a[], int n, int k)
{
   int left_end = partition (a, n);
   if (left_end+1==k) {
      return a[left_end];
   }
   if (left_end+1 > k) {
      return kth_smallest (_____);
   } else {
      return kth_smallest (_____);
   }
}
```

The missing arguments lists are respectively

A. (a,left_end,k) and (a+left_end+1,n-left_end-1,k-left_end-1)
B. (a,left_end,k) and (a,n-left_end-1,k-left_end-1)
C. (a+left_end+1,n-left_end-1,k-left_end-1) and (a,left_end,k)
D. (a,n-left_end-1,k-left_end-1) and (a,left_end,k)

Answer: A.

We have to find the k^{th} smallest element.

if (left_end+1 > k)

If the above condition is true, it means we have k^{th} smallest element on the left array, whose size is left_end instead of n for the original array. (The "+1" is used because the array index in C language starts from 0). So, we can do a recursive call as

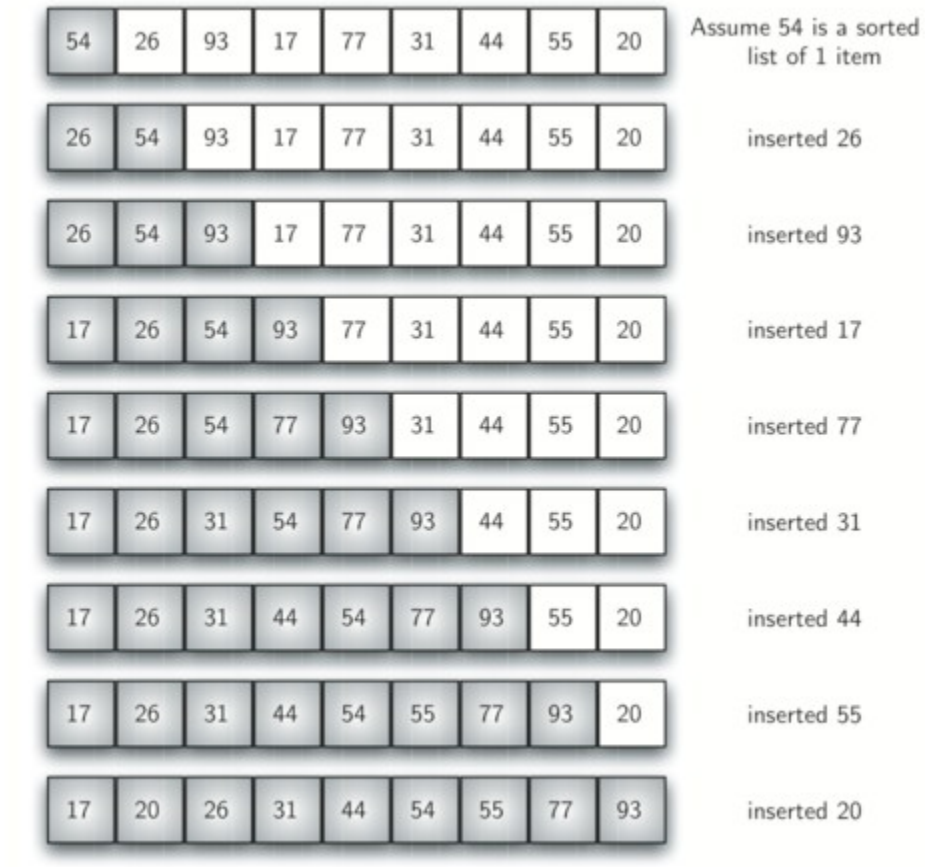
kth_smallest(a, left_end, k);

If the above condition is false, and $left\_end+1 \neq k$, it means k^{th} smallest element is on the right part of the array and it will be $(k-left\_end-1)^{th}$ element there as left_end+1 elements are gone in the left part. So, the recursive call will be kth_smallest

(a + left_end + 1, n - left_end – 1, k - left_end - 1);

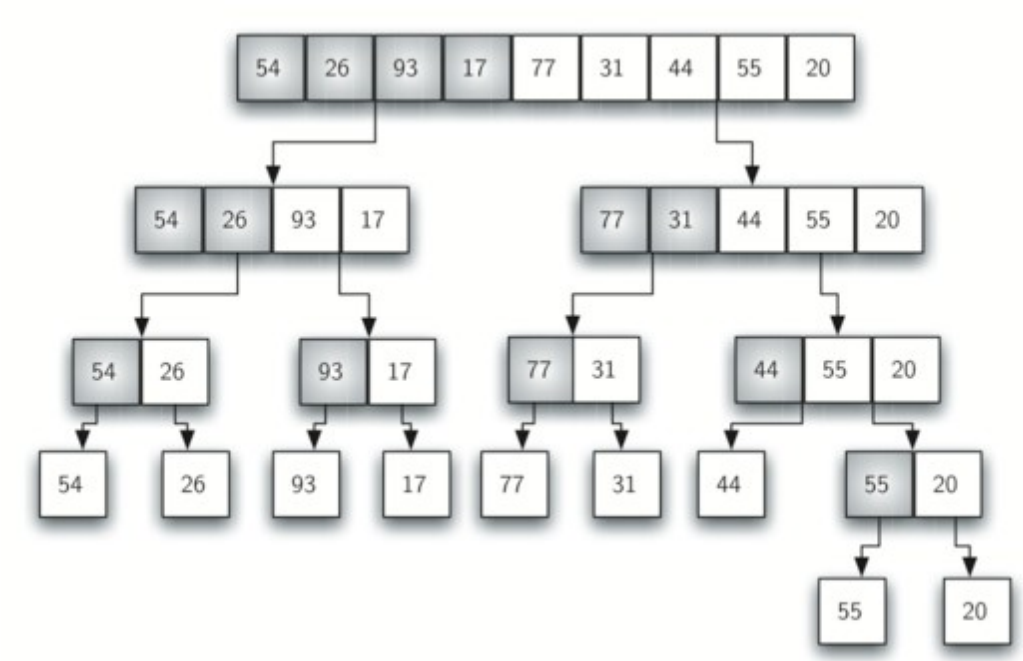
Q2. Demonstrate the insertion sort results for each insertion for the following initial array of elements . 54 26 93 17 77 31 44 55 20

Ans 2.

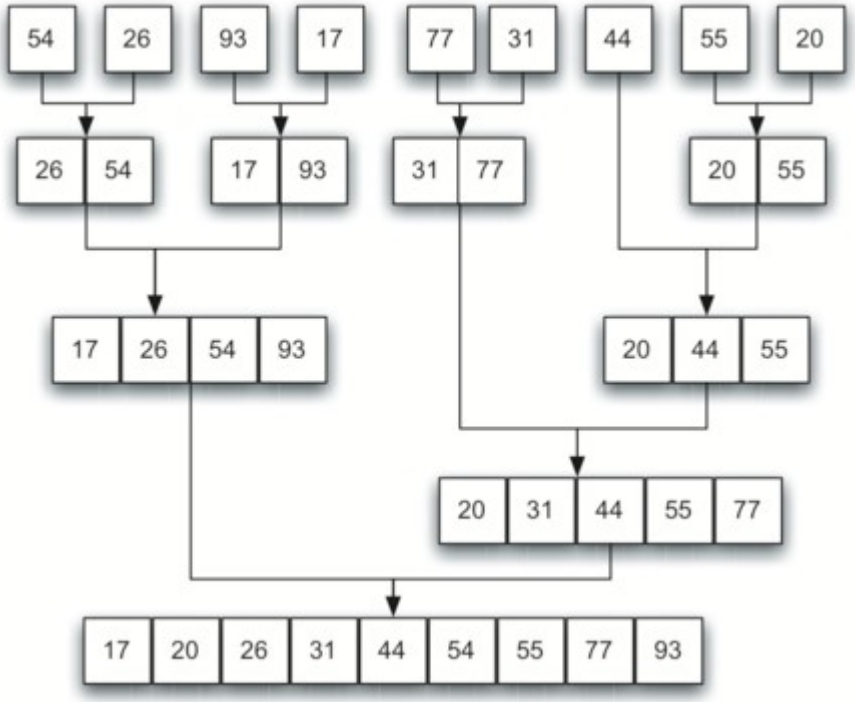


Q2. Demonstrate the insertion sort results for each insertion for the following initial array of elements . 54 26 93 17 77 31 44 55 20

Ans2.



Splitting the image



Q3. Suppose we are sorting an array of eight integers using quicksort, and we have just finished the first partitioning with the array looking like this:

2 5 1 7 9 12 11 10

Then what will be the pivot number?

Ans 3. 7 and 9 both are at their correct positions (as in a sorted array). Also, all elements on left of 7 and 9 are smaller than 7 and 9 respectively and on right are greater than 7 and 9 respectively.

Q5. Apply Quick sort on a given sequence **7 11 14 6 9 4 3 12**. What is the sequence after first phase, pivot is first element?

Ans: 6 3 4 7 9 14 11 12
Explanation: Let's apply Quick sort on the given sequence,
For first phase, pivot = 7

7	11	14	6	9	4	3	12
	i						**j**
7	11	14	6	9	4	3	12
	i					**j**	
7	3	14	6	9	4	11	12
		i			**j**		
7	3	4	6	9	14	11	12
			i	**j**			
7	3	4	6	9	14	11	12
			j	**i**			
6	3	4	7	9	14	11	12

Structures and Unions
Q1 What will be the output of the following program.

```
#include‹stdio.h›
int main()
{
   struct site
   {
      char name[] = "GeeksQuiz";
      int no_of_pages = 200;
   };
   struct site *ptr;
   printf("%d ", ptr->no_of_pages);
   printf("%s", ptr->name);
   getchar();
   return 0;
}
```

Ans1: Compiler Error

When we declare a structure or union, we actually declare a new data type suitable for our purpose. So we cannot initialize values as it is not a variable declaration but a data type declaration.

Q2. If the size of int is 32 bit (4 byte) then what will be the output of the following problem.

```
#include<stdio.h>
struct st
{
   int x;
   static int y;
};
int main()
{
   printf("%d", sizeof(struct st));
   return 0;
}
```

Ans2. In C, struct and union types cannot have static members.

Q3. Assume that objects of the type short, float and long occupy 2 bytes, 4 bytes and 8 bytes, respectively. (Gate CS 2000)

```
struct {
    short s[5];
    union {
        float y;
        long z;
    }u;
} t;
```

Ans 3: Short array s[5] will take 10 bytes as size of short is 2 bytes.
When we declare a union, memory allocated for the union is equal to memory needed for the largest member of it, and all members share this same memory space. Since u is a union, memory allocated to u will be max of float y(4 bytes) and long z(8 bytes). So, the total size will be 18 bytes (10 + 8).

Q4. What will be the output of the following

```
# include <stdio.h>
# include <string.h>
struct Test
{
char str[20];
};
```

```
int main()
{
struct Test st1, st2;
strcpy(st1.str, "GeeksQuiz");
st2 = st1;
st1.str[0] = 'S';
printf(st2.str);
return 0;
}
```

Ans 4. Array members are deeply copied when a struct variable is assigned to another one. It means it cannot be edited.

Pointers

Q1) What is printed by the following C program?

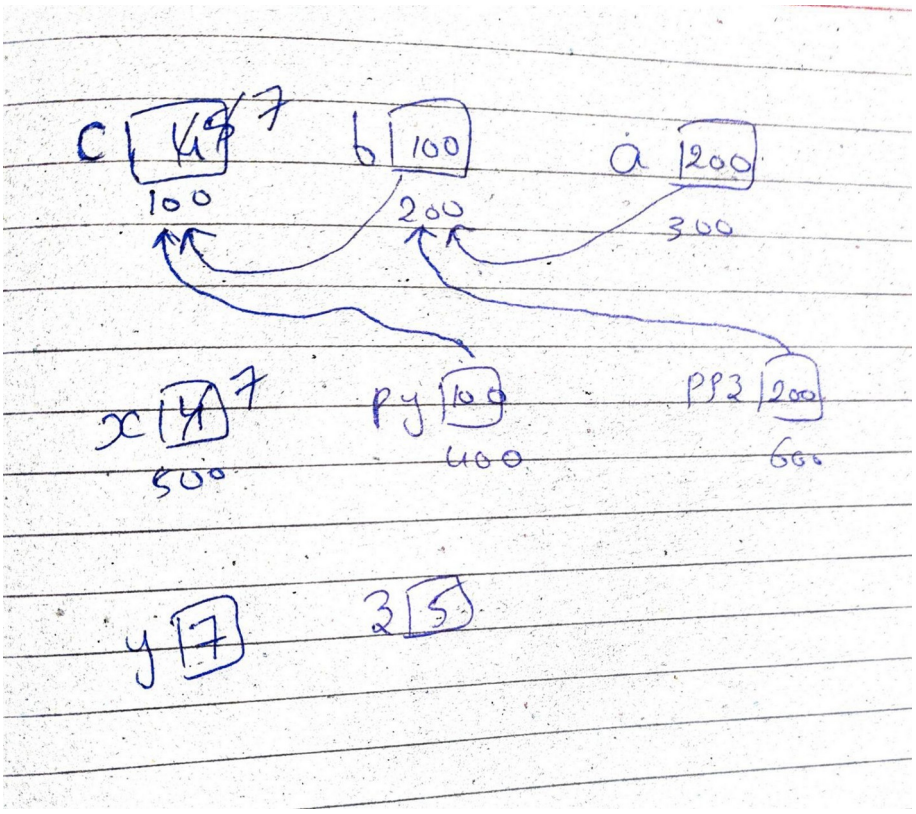
```
int f(int x, int *py, int **ppz)
{
     int y,z;
     **ppz+=1; z=**ppz;
    *py+=2; y=*py;
     x+=3;
     return x+y+z;
}
void main()
{
    int c, *b, **a;
    c=4; b=&c; a=&b;
    printf("%d",f(c,b,a));
}
```

a)18
b)19
c)21
d)22

Answer: b

Solution:

The below figure shows what is pointing to what



x+y+z is asked which is 7+7+5=19.

Q2) What is the output of the following program?

```
#include<stdio.h>
int main()
{
   int a[ ][3]= {1,2,3,4,5,6};
   int (*ptr)[3]=a;
   printf("%d %d" , (*ptr)[1], (*ptr)[2]);
   ++ptr;
   printf("%d %d", (*ptr)[1], (*ptr)[2]);
   return 0;
}
```

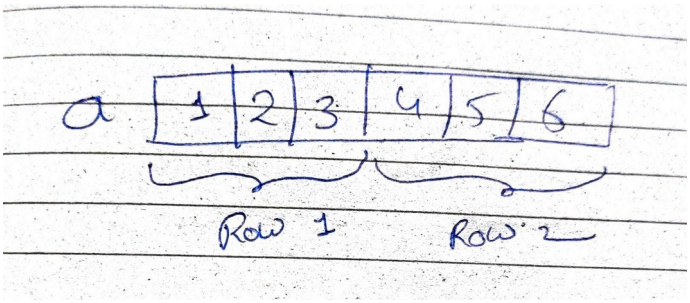
a)2 3 5 6
b)2 3 4 5
c)4 5 0 0
d)None of the above

Answer: a

Solution:

ptr is pointer to first 1D array.

*ptr is pointer to first element of first 1D array.



(*ptr)[1] can be written as *((*ptr)+1)
So, (*ptr) + 1 means you point to second element of first 1D array and dereferencing it by *((*ptr)+1) will give you value 2.

Similarly, (*ptr)[2] can be written as *((*ptr)+2)
So, (*ptr) + 2 means you point to third element of first 1D array and dereferencing it by *((*ptr) +2) will give you value 3.

++ptr will point to the second 1D array ,ie, row 2.

(*ptr)[1] which is equal to *((*ptr)+1) will fetch second element from the second 1D array which is 5.

Similarly, (*ptr)[2] will fetch third element from the second 1D array which is 6.

So, the final answer is 2 3 5 6.

Q3: What will be the output of the following code

```
void fun(int *p) {
  int q = 10;
  p = &q;
}

int main() {
```

```
  int r = 20;
  int *p = &r;
  fun(p);
  printf("%d", *p);
  return 0;
}
```

Ans: **20**

Explanation:

When passing p to fun(), we pass a copy of p so changing its value inside fun() does not affect its value in main. If we want to change a local pointer of one function inside another function, then we must pass a pointer to the pointer. By passing the pointer to the pointer, we can change pointer to point to something else. See the following program as an example.

```
void fun(int **pptr) {
  static int q = 10;
  *pptr = &q;
}

int main() {
  int r = 20;
  int *p = &r;
  fun(&p);
  printf("%d", *p);
  return 0;
}
```

Q4) What is the value printed by the following C program.

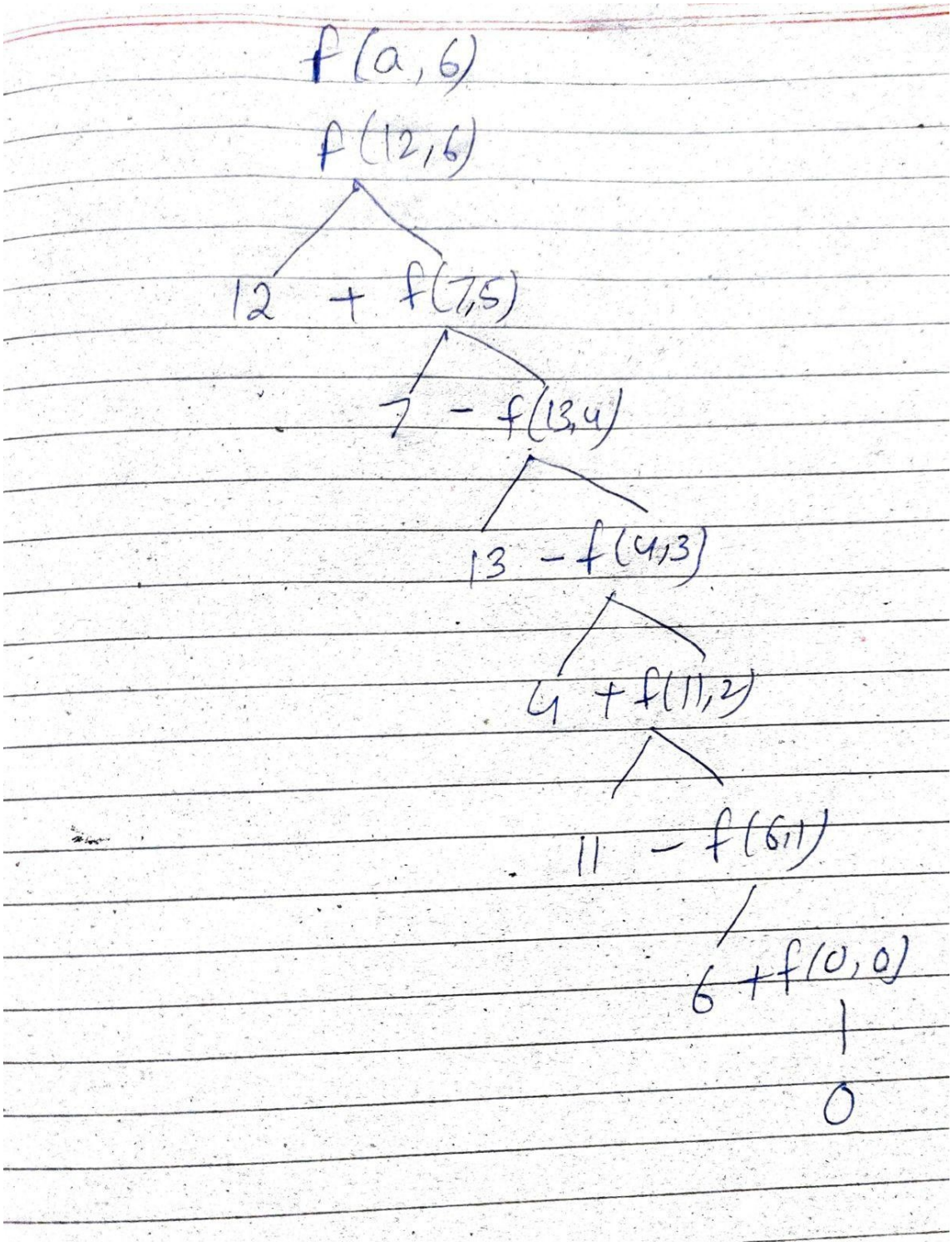
```
#include<stdio.h>
int f(int *a, int n)
{
   if(n <= 0) return 0;
   else if(*a % 2 == 0) return *a + f(a+1, n-1);
   else return *a - f(a+1,n-1);
}

int main()
{
   int a[ ]={12,7,13,4,11,6};
   printf("%d", f(a,6));
   getchar();
   return 0;
}
```

a)-9
b)5
c)15
d)19

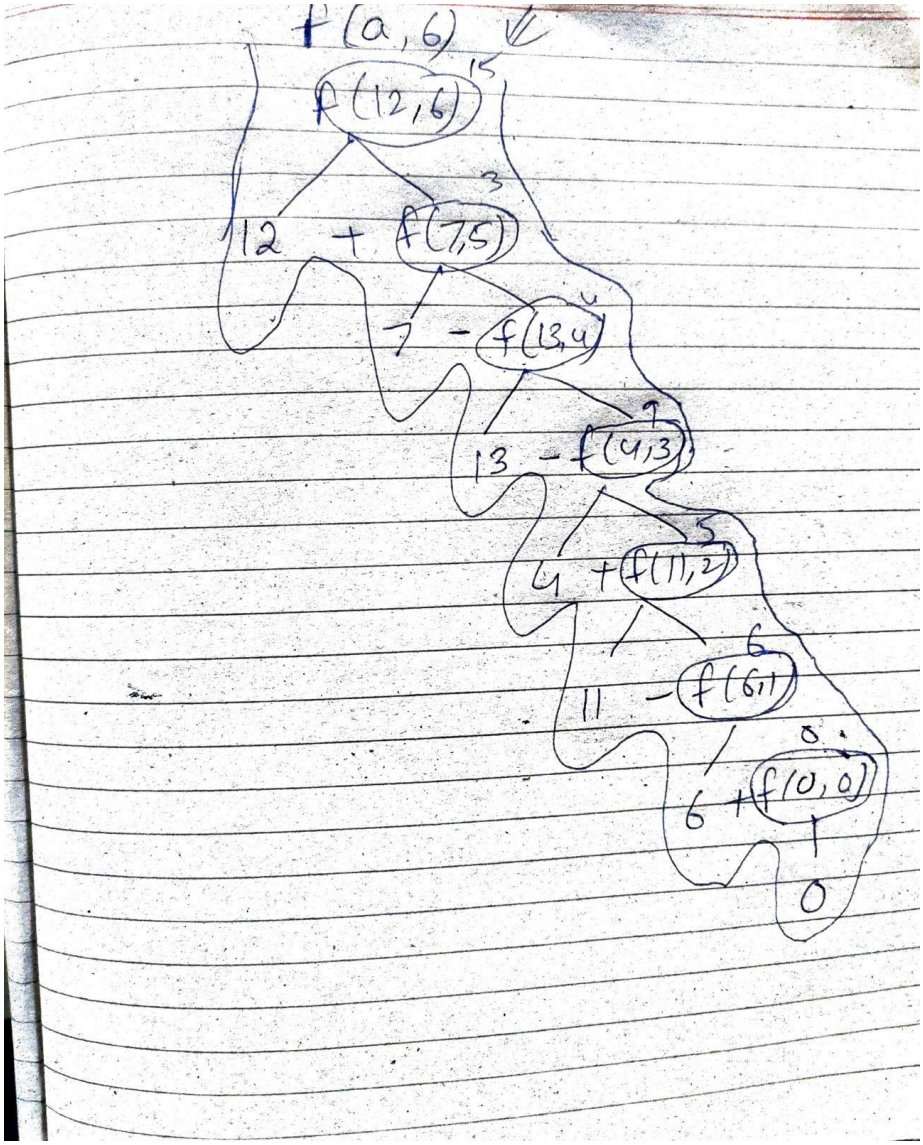
Answer: c

Explanation: _

Recursion is used in this question and we recursively call by adding when even number and subtracting when we get odd number till the base case when n becomes 0, we simply return 0.



Now, let's see the traversing of the above.



The final answer of f(a,6) is 15.

Q5)

```
#include <stdio.h>
int main ()
{
   int  a[4][5] = {{1, 2, 3, 4, 5},
                   {6, 7, 8, 9, 10},
                   {11, 12, 13, 14, 15},
                   {16, 17, 18, 19, 20}};
   printf("%d\n", *(*(a+**a+2)+3));
   return(0);
```

}

The output of the above program is ?

The answer is 19.

Explanation:

Let’s assume the base address to be 1000.

=> *(*(a+**a+2)+3))
=> *(*(1000+**1000+2)+3))
=> *(*(1000+3)+3)) {given element **1002 = 3}
=> *(*(1003)+3))
=> *((1003)+3) {4th row selected in given matrix}
=> *((1003)+3) {address of 4th element in 4th row}
=> a[3][3]
=> 19 {element selected a[3][3] = 19}