Q1
2 Points

Give all pairs of x, y values for which the following conditional expression evaluate to True, where both x and y are integers in the set $\{1, 2, 3, 4, 5\}$
:

$$(x + y == 6) \;||\; (x - y == 1)$$

(1,5) , (5,1) , (2,4) , (4,2) , (3,3) , (2,1) , (3,2) , (4,3) , (5,4)

Q2
5 Points

Professor X wants to write a C function that when given an integer array and its size as input, returns the number of non-zero integers present in the array. For some superstitious reason, he does not want to use any square brackets (i.e., [or]) in his function (including the function parameters). Professor X has already written the main function. You are asked to complete the function `countnz` for Professor X.

(*For the example array given in the* `main` *function below, the* `countnz` *function should return* 9.)

```
#include<stdio.h>

//This is the function you need to complete (including the input parameters)
//without using any square brakets
int countnz(                   ){




}

int main(){
    int A[] = {2, 34, 1, 67, 32, 0, 4, 0, 0, 5, 23, 0, 9, 0};
    int n = 14;

    printf("The number of non-zero elements are %d\n", countnz(A, n));
}
```

```
int countnz(int *A , int n){
// using a pointer and dereferncing to access the elements of the
array
   int cnt = 0;
   for( int i = 0 ; i<n ; i ++ ){
      if(*(A+i)!=0) cnt ++ ;
   }

   return cnt ;
}
```

Q3

6 Points

Professor X wants to write a C program that finds the largest 10 numbers in a given integer array containing distinct integers. He has written the main function and has also provided a `swap` function that you may use. For some superstitious reasons, Professor X does not want you to declare any new variable within the body of the function `insert`. He also does not want you to declare any global variables. You are asked to complete the `insert` function so that the `topten` array contains the largest 10 integers of the array `A` after the second for-loop in the `main` function. Note that the program then prints the largest 10 numbers in the last for-loop of the `main` function.

```c
#include<stdio.h>

void swap(int *x, int *y){
    int temp = *x; *x = *y; *y = temp;
}

//This is the function that you have to complete
//without declaring any new variables.
void insert(int topten[], int number, int j){



}

int main(){
    int A[] = {1, 56, 23, 43, 78, 54, 7, 21, 49, 77, 90,
                94, 25, 12, 45, 13, 67, 62, 46, 22};
    int n = 20;
    int i, topten[10];

    /* initialize all the elements of topten to -1 */
    for (i = 0; i < 10; i++){topten[i] = -1;}

    /* scan through the numbers array, just once */
    for (i = 0; i < n; i++){insert(topten, A[i], 0);}

    /* print the elements */
    for (i = 0; i < 10; i++){printf("%d\n",topten[i]);}
}
```

```
void insert( int topten[] , int number , int j ){
// maintaining the array topten in such a way that it always contains the largest 10 elements //in increasing order
   if(number>topten[j]){
//if the current number to be inserted is greater than the smallest of the current topten
// we replace it
      topten[j] = number ;
   }
// now we keep swapping till the element reaches its correct position and
// topten[0] again contains the smallest of the current  10 largest elements
   while(j<=8 && topten[j]>topten[j+1] ){
      swap(&topten[j] , &topten[j+1]) ;
      j ++ ;
   }
```

return ;
}

Q4
10 Points

Consider the following C functions and answer the questions that follow. *In some of the questions, you are asked to give a count on the number of function calls. Please do precise counting (do NOT ignore additive or multiplicative constants).*

```
#include<stdio.h>

int compare(int x, int y){
    if(x <= y)return 1;
    else return 0;
}

void func(int A[], int n){
    int x, y;
    if(compare(A[0], A[1])){x = A[0]; y = A[1];}
    else {x = A[1]; y = A[0];}

    for(int i =2 ; i<n; ++i){
        if(compare(A[i], x)){y=x; x = A[i];}
        else if(compare(A[i], y))y = A[i];
    }
    printf("%d, %d\n", x, y);
}
```

Q4.1
2 Points

In one sentence clearly state what the function `func` does when given an integer array `A` and array size `n` as input. (*You need to figure out what the function outputs and NOT just rewrite the description of the code.*)

The given function func outputs the smallest number in the array followed by a space and then the second smallest element in the array A it maintains the smallest number in x and the second smallest number in y and keeps iterating through the elements of the array and changing them accordingly and eventually outputs the smallest and the second smallest elements in the array.

Q4.2
2 Points

How many calls to the function `compare` is made when `func(A, n)` is called with `A` and `n` declared as:
`int A[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}; int n = 10`?

9

One call is made in the beginning before entering the loop. In every iteration one call is made for the first if statement and since this always returns true since always A[i] < x. Hence the total number of calls is 1 + 8 (the number of times the loop runs is 9-2+1)

Q4.3
2 Points

What is the worst case number of calls to the function `compare` that is made when the function `func` is called on an array of size 10? Note that the worst case is across all integer arrays of size 10.

17
The maximum number of calls occur when the first if statement evaluates to false and hence in every iteration the else if statement is also checked and hence two calls are made to the compare function in every iteration of the loop hence the number of maximum calls, is 1 (for the call before the loop)+ 8*2 (two calls in each of the 8 iterations of the loop)

Q4.4
2 Points

Give an integer array of size 10 on which maximum number of calls to `compare` is made from `func`. In other words, give a worst-case array for the previous part.

A[10] = {1 , 10 , 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 }

Here since x = 1 and always is <=A[i] the first if statement evaluates to false and in each iteration of the loop two calls are made

Q4.5
2 Points

What is the worst-case number of calls to the function `compare` that is made when the function `func` is called on an array of size n in general. Your answer should be a function of n. You may assume that $n \geq 2$.

The number of worst-case calls are 2*n-3 for an array of size n
One call is always made in the starting
the loop runs for i=2 to i=n-1 in every iteration a maximum of two calls can be made which happens when the first if statement evaluates to false this can happen if x stores the minimum element in the entire array before coming into the loop itself hence A[i]<= x will always give false if A[i]>x for i=2 to i=n-1.
The total number of calls are 1 + 2*(n-2) = 2n-3

Q5
6 Points

Forgetful Professor X wrote the following recursive C program but sometime later forgot what the program does. You are asked to help him figure out what the program does. Answer the questions that follow:

```c
#include<stdio.h>

int func2(int n, int v){
    if(n == 0)return v;
    if(n%2 == 0)return func2(n/2, v);
    return func2(n/2, v+1);
}

int main(){
    int n;
    printf("Enter a positive number: ");
    scanf("%d", &n);

    printf("%d\n", func2(n, 0));
}
```

Q5.1
2 Points

What does the program print when the user enters the integer 25?

3

The following recursive calls are made
(25 , 0) => (12 , 1) => (6 , 1) => (3 , 1) => (1 , 2) => (0 , 3)   The last call returns 3 and this 3 is returned backwards subsequently by all function calls and eventually the first call made to func2().

Q5.2
2 Points

What the program print when the user enters a positive integer whose binary representation is `10011100011010101010`?

10
For the given binary number the answer will be 10 because 10 bits are set to 1. If n is odd in the current function call (last bit set to one) then v increases for the next function call otherwise if it is even then v remains the same for the next function call n/2 right shifts the current value of n by one which drops the last bit and moves to second last bit until the number becomes 0

Q5.3
2 Points

Write one sentence that clearly states what the program outputs when given a positive integer n as input. (*Do NOT simply rewrite the description of the code.*)

The above program basically counts the number of bits set to one in the binary representation of the number because if the number is odd then the last bit is one and v increases by one if the number is even then the last bit is 0 and v doesn't change for the next call .
Doing n/2 has the effect of right shifting the current value of n by one which drops the last digit and checks for the next digit until we hit the

last set bit This way the program counts the number of bits set to one.

Q6

6 Points

Write a C function that when given an integer array `A` and its size `n` outputs the number of distinct integers present in the array. You may assume that the input array `A` contains numbers between 1 and 1000. We have given the main function so that you can focus just on the function.

(*For the example array* `A` *given in the* `main` *function, the number of distinct elements present is* 12.)

```c
#include<stdio.h>

//This is the function that your need to complete
int distinct(int A[], int n){



}

int main(){
    int A[] = {1, 456, 12, 432, 54, 67, 888, 23, 1, 43, 65, 78, 34, 1, 1};
    int n = 15;

    printf("The number of distinct integers in the array is: %d\n", distinct(A, n));
}
```

```
int distinct( int A[] , int n ){
   int frequency[1001] ;
   for(int i = 0 ;i<=1000 ; i ++ ){
      frequency[i] = 0 ;
   }
   for( int i = 0 ; i<n ; i ++ ){
      frequency[A[i]] ++ ;
   }
   int ans = 0 ;
   for( int i = 0 ; i<=1000 ; i++ ){
      if(frequency[i]) ans ++ ;
   }
```

```
    return ans ;
}
```

Major

● Graded

Student

Chinmay Mittal

View or edit group

Total Points

35 / 35 pts

Question 1		
(no title)		**2** / 2 pts
Question 2		
(no title)		**5** / 5 pts
Question 3		
(no title)		**6** / 6 pts
Question 4		
(no title)		**10** / 10 pts
4.1	(no title)	**2** / 2 pts
4.2	(no title)	**2** / 2 pts
4.3	(no title)	**2** / 2 pts
4.4	(no title)	**2** / 2 pts
4.5	(no title)	**2** / 2 pts
Question 5		
(no title)		**6** / 6 pts
5.1	(no title)	**2** / 2 pts

5.2	(no title)	**2** / 2 pts
5.3	(no title)	**2** / 2 pts

Question 6

(no title)	**6** / 6 pts