CS Assig

Ans. to the Q. No.-1

a) here the third loop iterates infinite times.

The runtime is infinity.

b) there is two block of code here. The later one runs infinite times. The runtime is infinity.

Ans. to the Q. No. -2

a) def binary (arr, num):

```
    l, r = 0, length of arr
    index=0
    while  l <= r:
        mid = (l+r)//2
        if  arr[mid] == num:
            index=mid
            right = mid-1

        elif  arr[mid] > num:
            right = mid-1

        else:
            left = mid +1
    end while
    return index
```

endfunc

b)
```
def binary(arr, num, leftsearch):
    l,r = 0, len(arr)-1
    idx=0
    while l<=r:
        mid = (l+r)//2
        if arr[mid] == num:
            idx = mid
            if leftsearch:
                right = mid-1
            else:
                left = mid+1
        elif arr[mid] > num:
            right = mid-1
        else:
            left = mid+1
    end while
    return idx
end func
print(binary(arr, num, True), binary(arr, num, True)-binary(
                                    arr, num, False)
```

Ans. to the Q. No. -3

T = 2

Step	L	R	m
1	0	7	3
2	0	2	1

Even though the list is not sorted the binary search still works. As the first A[mid] is bigger than 2 and the next A[mid] is 2.

Ans. to the Q. No. 4

a)
```
def func(arr):
    l,r = 0, len(arr)-1
    while l<=r:
        mid = (l+r)//2
        if (mid==0 or arr[mid]> arr[mid-1]) and
          (mid == len(arr)-1 or arr[mid+1]<arr[mid]):
            return arr[mid]
        elif arr[mid] < arr[mid+1]:
            left = mid+1
        else:
            right = mid-1
    end while
end func
```

b) every time the process is getting halfed in this algorithm.

$$n \to \frac{n}{2} \to \frac{n}{4} \cdots \to 1$$

$$\therefore 1 = \frac{n}{2^k}$$

$$\Rightarrow 2^k = n$$

$$\Rightarrow k = \log_2 n$$

$$\therefore O(\log n)$$

Ans. to the Q. No. -5

a) def func(num):

```
    i = 1
    while i*i <= n:
        i += 1
    end while
    return i-1
end func
```

b) def func(num):

```
    l, r = 1, num
    while l <= r:
        m = (l+r) // 2
        if m*m == num:
            return m
        elif m*m > n:
            r = m-1
        else:
            l = m+1
    end while
    return r
endfunc
```

Ans. to the Q. No. - 6

a) It will be efficient when doing multiple searches.

b)
```
def csort(arr):
    m1 = min value of arr
    m2 = max value of arr
    count = (m2-m1+1) size array
    for i in arr:
        count[i-m1] += 1
    end for
    i = 0
    for j in range (m2-m1+1):
        while count[j] > 0:
            arr[j] = j + m1
            i += 1
            count[j] -= 1
        end while
    end for
    return arr
```

c)

```
def csort(arr):
    max_dec = highest number of digits before (.)
    scale = 10** max_dec
    arr = [ int(num * scale) for num in arr]
    m1 = min(arr)
    m2 = max(arr)
    count = [0] * (m2 - m1 + 1)
    for i in arr:
        count[i - m1] += 1
    end for
    i = 0
    for j in range (m2 - m1 + 1):
        while count[j] > 0:
            arr[i] = (j + m1) / scale
            i += 1
            count[j] -= 1
        end while
    end for
    return arr
end func
```

d) as I don't have much memory and merge sort needs more memory. So I'll use quick sort which uses inplace sorting.

e) [1, 2, 3, 4, 5]

Ans. to the Q. No:- 7

a)
```
def sort(arr):
    i = 1
    j = len(arr) - 2
    new = [ ]
    while i < len(arr) and j > -1:
        if arr[i] > arr[j]:
            new.append(arr[j])
            j -= 2
        else:
            new.append(arr[i])
            i += 2
    end while
```

```
while i< len(arr):
    new.append (arr[i])
    i+=2
endwhile
while j>-1:
    new.append(arr[j])
    j-=2
end while
return new
end func
```