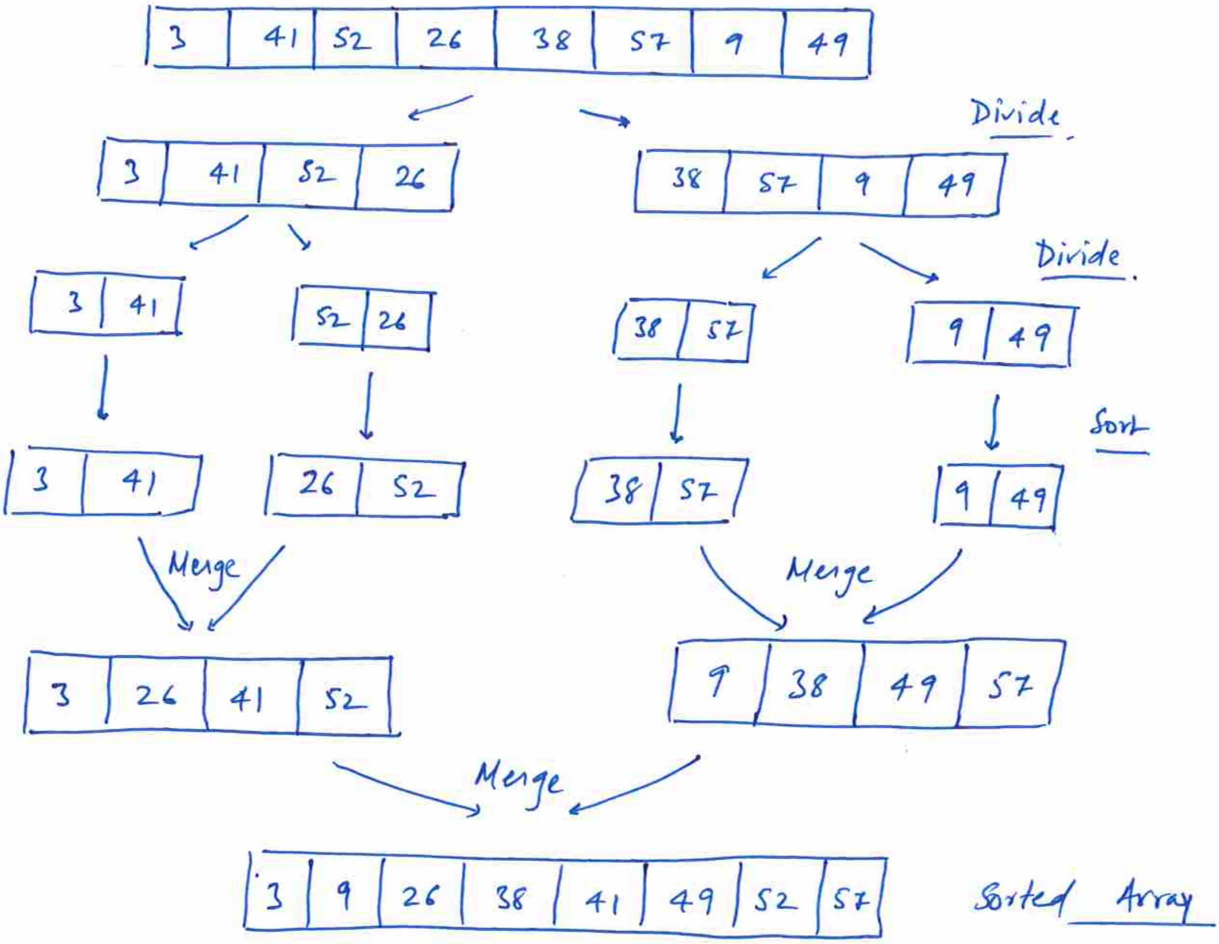
Qn 2.3-1

$A = \langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$.



Qn 2.3-2

```
Merge ( A, p, q, r )

n1 = q - p + 1
n2 = r - q
for i = 1 to n1
        L[i] = A[p+i-1]
for j = 1 to n2
        R[i] = A[q+j]
i = 1 ;    j = 1 ;   k = p
while ( i < n1  && j < n2 )      // Limited copying using n1 and n
        if  L[i] <= R[j]         //.        instead of 'sentinels.'
              A[k] = L[i]
              i ← i+1
        else
              A[k] = R[j]
              j ← j+1
        k ← k+1
while  i < n1                    // All (left) L elements copied back.
        A[k] = L[i]
        i ← i+1 ; k ← k+1
while  j < n2                    // All (left) R elements copied back.
        A[k] = R[j]
        j ← j+1 ; k ← k+1
```

Qn 2.3-3

$$T(n) = \begin{cases} 2 & \text{if } n = 2, \\ 2T(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

is $T(n) = n \log(n)$?

When n is an exact power of 2

$\Rightarrow$ $n = 2^k$ for some $\cancel{k} > 1$.

$$\Rightarrow \quad T(n) = 2\,T(n/2) + n. \qquad \& \qquad T(2) = 2.$$

$$= 2\left(2\,T(n/4) + n\right) + n$$

$$= \cancel{2}\,4\,T(n/4) + 3n$$

$$= 4\left(2\,T(n/8) + n\right) + 3n$$

$$= 8\,T(n/8) + \cancel{4n}\ 7n$$

$$\vdots$$

$$= 2^k\,T\left(n/2^k\right) + \left(2^k - 1\right) n.$$

For $T(n/2^k) = T(2) \Rightarrow \frac{n}{2^k} = 2 \Rightarrow n = 2^{k+1}$

or $k+1 = \log n$

or $k = (\log n - 1$

$$T(n) \le 2^{(\log n - 1)} .\ T(2) + \left(2^{\log n - 1} - 1\right) n.$$

$$\le 2^{\log n} + \frac{2^{\log n}}{2} . n - n.$$

$$\le \log n + \frac{n \log n}{2} - n$$

$$= O(n \log n).$$

Qn 2.3 -4

```
                    T(n).
* insertionSort ( A, n )        // Sorts Array A[1...n].

  if n = 1 then                                           } c .
      return A[1]
  else
      insert ( A[n], insertionSort(A, n-1), n )    →  T(n-1) ~~+c~~


* insert ( key, A, n )     // inserts 'key' into ~~a~~ sorted A[1...n-1]

  for i = n-1 to 1
      if key < A[i] then.
          A[i+1] = A[i]
      else
          ~~A[i+1] = key~~
          break;
  A[i+1] = key:
```

<u>Proof</u>: (By induction) on n. (For recursive InsertionSort (A, n)).

<u>THEOREM</u>: InsertionSort(A, n) correctly sorts A[1...n]

<u>Base</u>: $n = 1$ – array has only one element
⇒ element already sorted
– returns A[n] (returns the only element).

<u>Step</u>: $n > 1$

IH: InsertionSort works correctly on less than n elements.

Recursive Call: insertionSort(A, n-1); $n-1 < n$: Always.

Size of problem = $n-1$ ~~+1~~ $< n$

By IH it correctly sorts A[1...n-1]

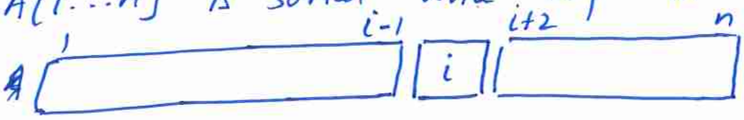
RT <u>Analysis</u>: $T(n) = T(n-1) + c$; $T(1) = 1$ $n - k = 1$

$\Rightarrow = T(n-k) + kc \leq \Theta(n)$ $\Rightarrow (k = n-1)$

Proof of insert(key, A, n) method : By Loop Invariant.

Post : A[1...n] is sorted with 'key' in correct place.

LI :



key < A[i+2 ... n]

A[1 .. i, i+2 ... n] contains what used to be in A[1...n-1]

A[i+1] is ~~the key~~. being checked.

Initialization : $i = n-1$

key < A[n+1 ... n] = ϕ ✓.

A[1 ... n-2, n] contains A[1 ... n-1]

A[n-1] is being checked.

Maintainance : ~~ac~~

~~key < A[i+2 ... n]~~

Only Case ~~1~~ : key < A[i]

⇒ A[i+1] = A[i]

key moves to check A[i-1]

key < A[i+1 ... n]

~~key~~ A[~~0~~ 1 ... i-2, i+1 ... n] ~~lies~~ has what was in A[1..n

~~Case 2:~~

Termination : Case 1 : ~~if~~ i = 0

A[i+1] = A[1] is set to key.

key = A[1] < A[2 ... n]

⇒ A[1 ... n] is sorted with key at A[1] ✓. (Post)

Case 2 : key > A[i]

⇒ loop breaks,

⇒ A[i+1] is set to key.

A[1] < A[2] ... < A[i] < key < A[i+2] ... < A[n]

⇒ A[1 .. n] is sorted, 'key' in correct place.

Qn 2.3-5

```
binary Search (A, key)  // returns index of key in A.

p = 0 ; r = n
~~for~~ while (p < r)
    q = ⌊(p + r) / 2⌋
    if A[q] < key then
        p = q + 1
    else if A[q] > key then
        r = q - 1
    else
        return q
```

```
recursive Binary Search (A, ~~key~~ p, r, key) // returns index of key in A
if (p < r).
    q = ⌊(p + r) / 2⌋
    if A[q] < key then
        return recursive Binary Search (A, q+1, r, key)
    else if A[q] > key then
        return recursive Binary Search (A, p, q-1, key)
    else
        return q
```

Qn 2.3-6

```
Insertion-Sort ( A ) ~~n~~).
for $j = 1$ to $n$.
    key = $A[j]$
    $p = 1$ ; ~~q~~ $r = j-1$
    while $p \nless r$
        $q = \lfloor (p+r)/2 \rfloor$
        if $A[q]$
```

It is possible to search the correct location for the element but moving the others one place up would take another linear time work which means the algorithm ~~a~~ will run in the order of $O(n \, . \, n\log n)$ for ~~a~~ worst case.

Qn 2.3-7

```
* sum(S, x)        // finds two elements in S having sum =
① for i = 1 to n              // n = |S|.   // Assuming the elements
②     key = x - S[i]                          are sorted
②     p = i+1                                 S.
①     r = n
①     while p < r
③         q = ⌊(p+r)/2⌋
③ ①       if S[q] > key then
  ②           p = q+1
          else if S[q] < key then
              r = q-1
          else
              return S[i], S[q].
      return False.
```

RT Analysis:

for loop runs n times.

$7\log(r-p+1)$

~~$7\log(n)$~~ $= 7\log(r-i)$

$$RT = \sum_{i=1}^{n} 7\log(r-i)+6$$

$= 6n + O(n(\text{max of } \Sigma))$

$= 6n + O(n\log(r-1))$

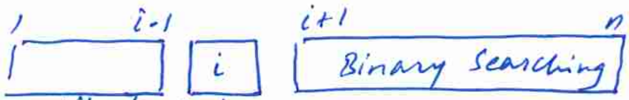
$= O(n\log n)$. $\{r = n$

Proof: by loop invariant.

Post: returns ~~true~~ two elements if their sum is x. otherwise returns false.

LI: for loop: None.

while loop:



- sorted.

⇐ key = sum − ith element.

key lies in $A[i+1 \dots n]$

And key $\notin A[i+1 \dots p-1]$ & key $\notin A[r+1 \dots n]$

$i+1 \le p \le r \le n$ → Not valid at termination.

for a sorted array, sum from a lower part is supposed to be found in upper part.

Initialization: