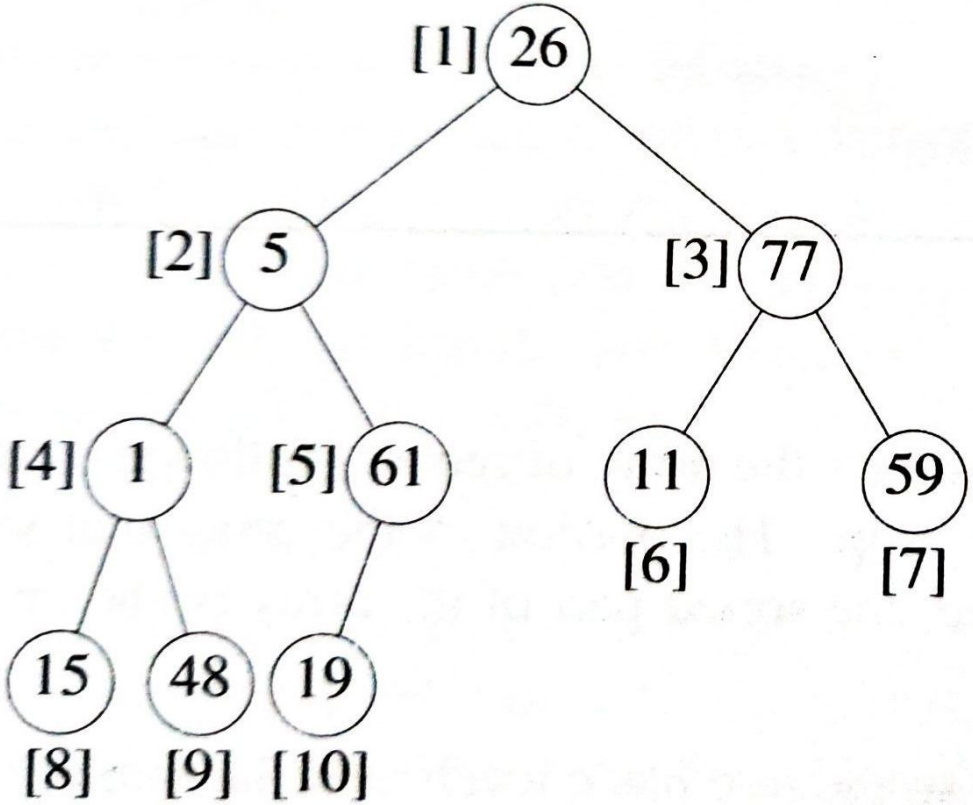
| $j$ | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|
| — | 5 | 4 | 3 | 2 | 1 |
| 2 | 4 | 5 | 3 | 2 | 1 |
| 3 | 3 | 4 | 5 | 2 | 1 |
| 4 | 2 | 3 | 4 | 5 | 1 |
| 5 | 1 | 2 | 3 | 4 | 5 |

| $j$ | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|
| — | 2 | 3 | 4 | 5 | 1 |
| 2 | 2 | 3 | 4 | 5 | 1 |
| 3 | 2 | 3 | 4 | 5 | 1 |
| 4 | 2 | 3 | 4 | 5 | 1 |
| 5 | 1 | 2 | 3 | 4 | 5 |

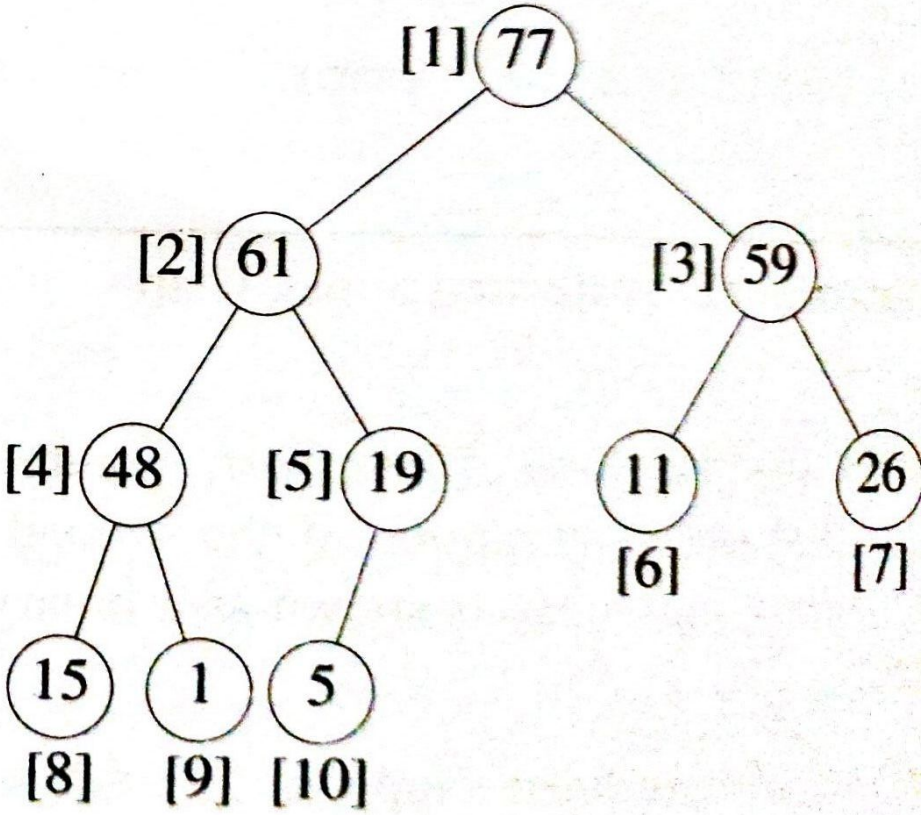
| $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $R_7$ | $R_8$ | $R_9$ | $R_{10}$ | *left* | *right* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| [26 | 5 | 37 | 1 | 61 | 11 | 59 | 15 | 48 | 19] | 1 | 10 |
| [11 | 5 | 19 | 1 | 15] | 26 | [59 | 61 | 48 | 37] | 1 | 5 |
| [ 1 | 5] | 11 | [19 | 15] | 26 | [59 | 61 | 48 | 37 | 1 | 2 |
| 1 | 5 | 11 | [19 | 15] | 26 | [59 | 61 | 48 | 37] | 4 | 5 |
| 1 | 5 | 11 | 15 | 19 | 26 | [59 | 61 | 48 | 37] | 7 | 10 |
| 1 | 5 | 11 | 15 | 19 | 26 | [48 | 37] | 59 | [61] | 7 | 8 |
| 1 | 5 | 11 | 15 | 19 | 26 | 37 | 48 | 59 | [61] | 10 | 10 |
| 1 | 5 | 11 | 15 | 19 | 26 | 37 | 48 | 59 | 61 | | |

```
void adjust(element a[], int root, int n)
{/* adjust the binary tree to establish the heap */
   int child,rootkey;
   element temp;
   temp = a[root];
   rootkey = a[root].key;
   child = 2 * root;                              /* left child */
   while (child <= n) {
      if ((child < n) &&
      (a[child].key < a[child+1].key))
         child++;
      if (rootkey > a[child].key) /* compare root and
                                     max. child */

         break;
      else {
         a[child / 2] = a[child]; /* move to parent */
         child *= 2;
      }
   }
   a[child/2] = temp;
}
```

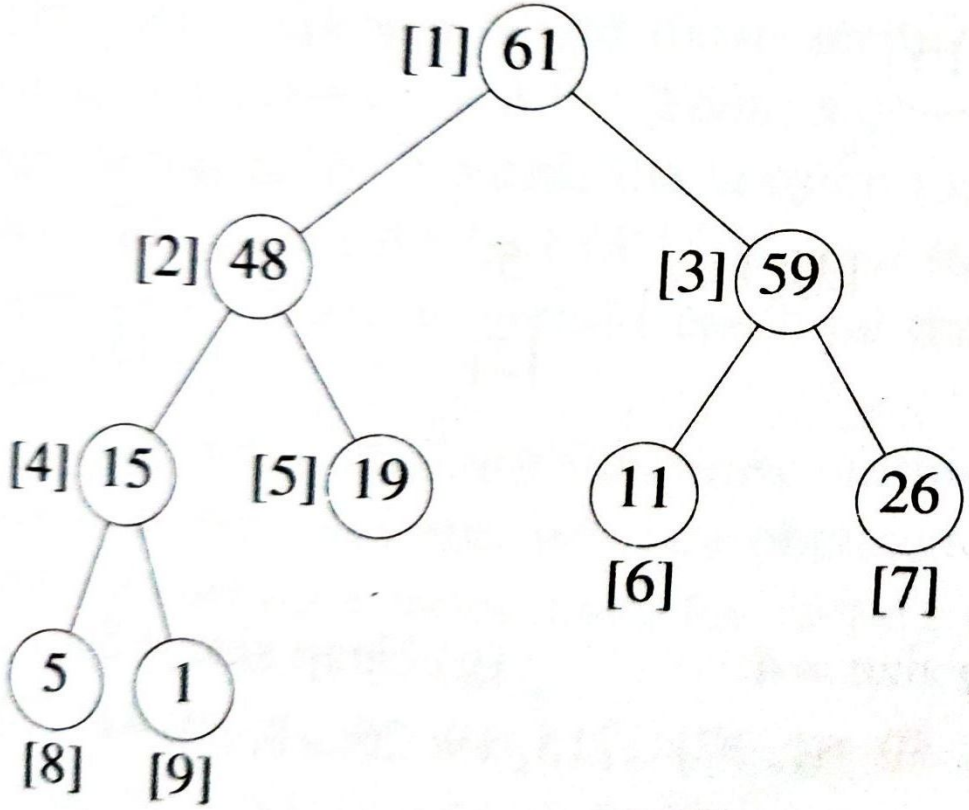
```
void heapSort(element a[], int n)
{/* perform a heap sort on a[1:n] */
   int i,j;
   element temp;

   for (i = n/2; i > 0; i--)
      adjust(a,i,n);
   for (i = n-1; i > 0; i--) {
      SWAP(a[1],a[i+1],temp);
      adjust(a,1,i);
   }
}
```




(a) Input array

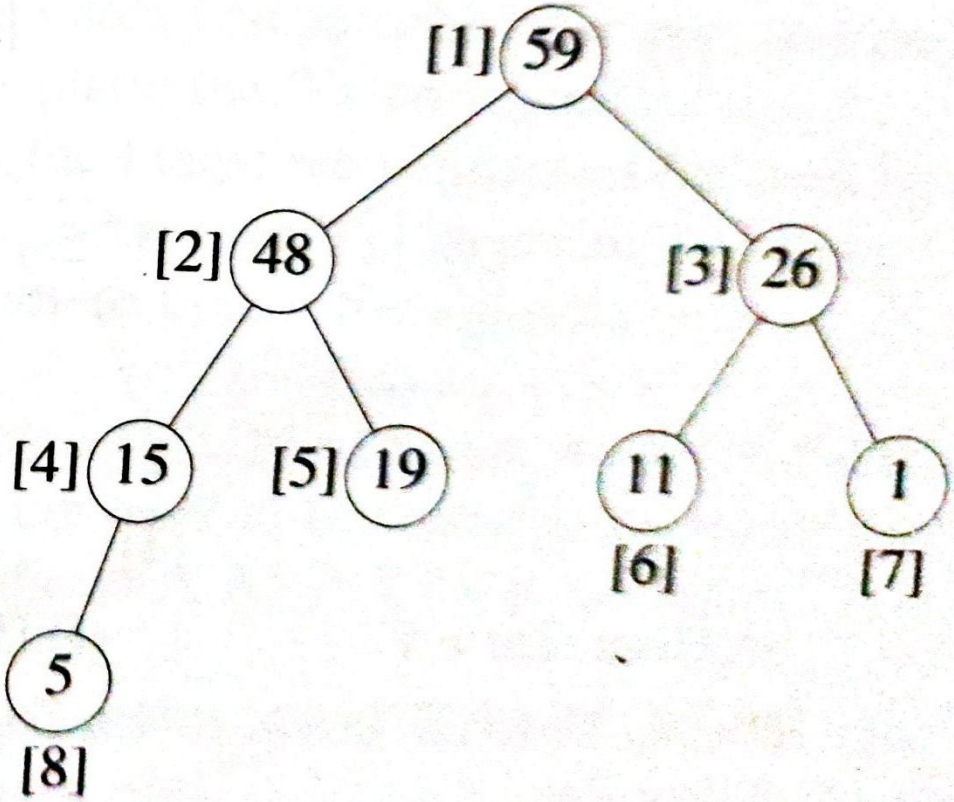



(b) Initial heap



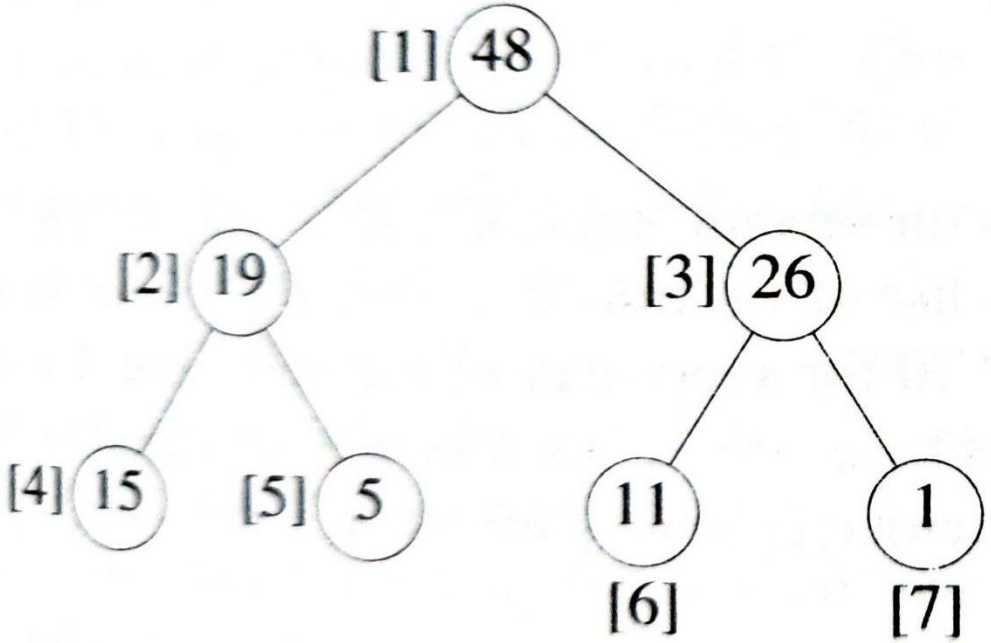

(a) Heap size = 9

Sorted = [77]



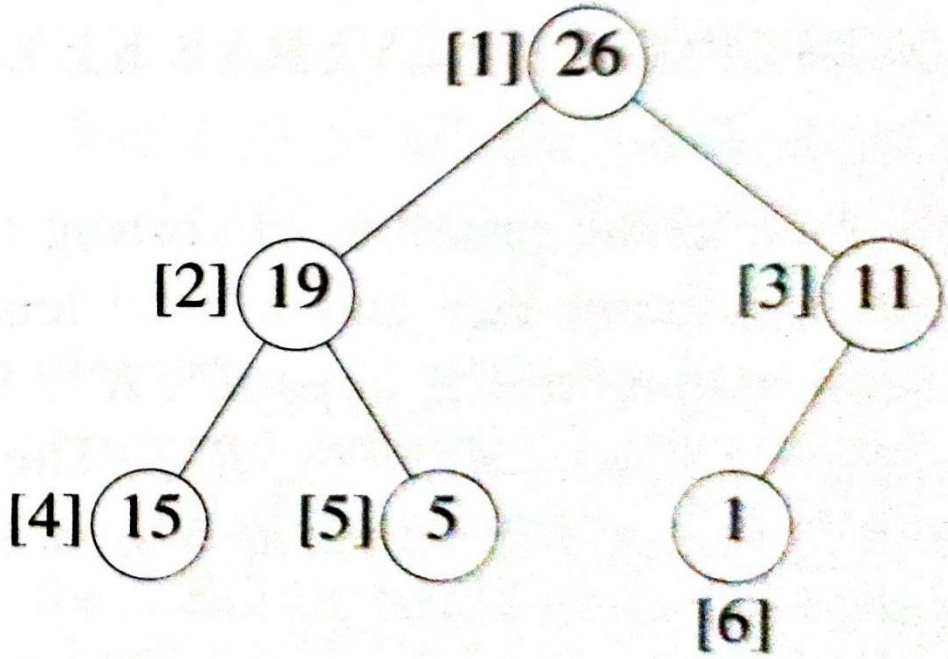

(b) Heap size = 8

Sorted = [61, 77]



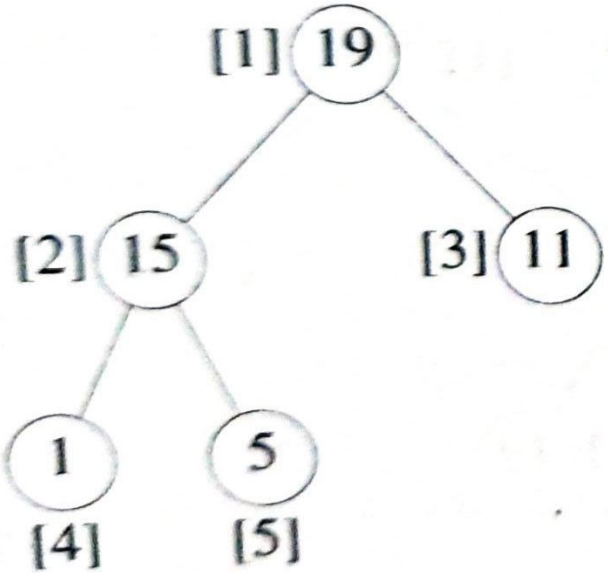

(c) Heap size = 7

Sorted = [59, 61, 77]



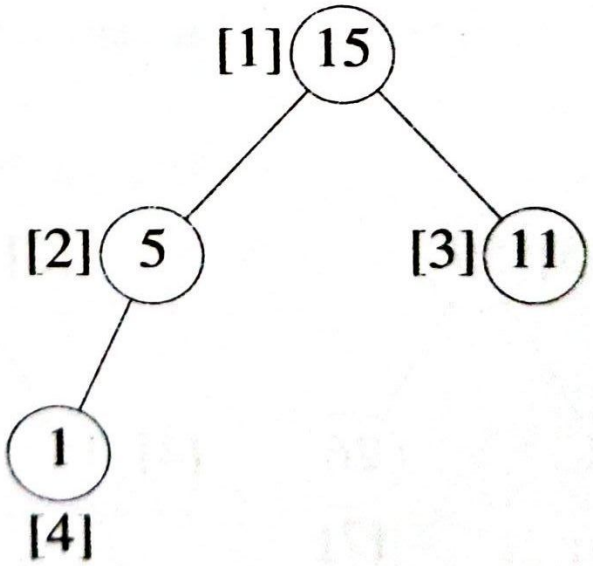

(d) Heap size = 6

Sorted = [48, 59, 61, 77]



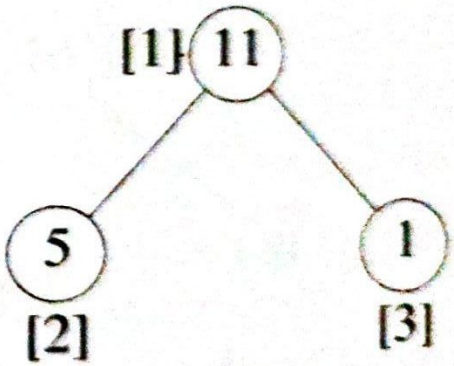

(e) Heap size = 5

[26, 48, 59, 61, 77]




(f) Heap size = 4

[19, 26, 48, 59, 61, 77]




(g) Heap size = 3

[15, 19, 26, 48, 59, 61, 77]

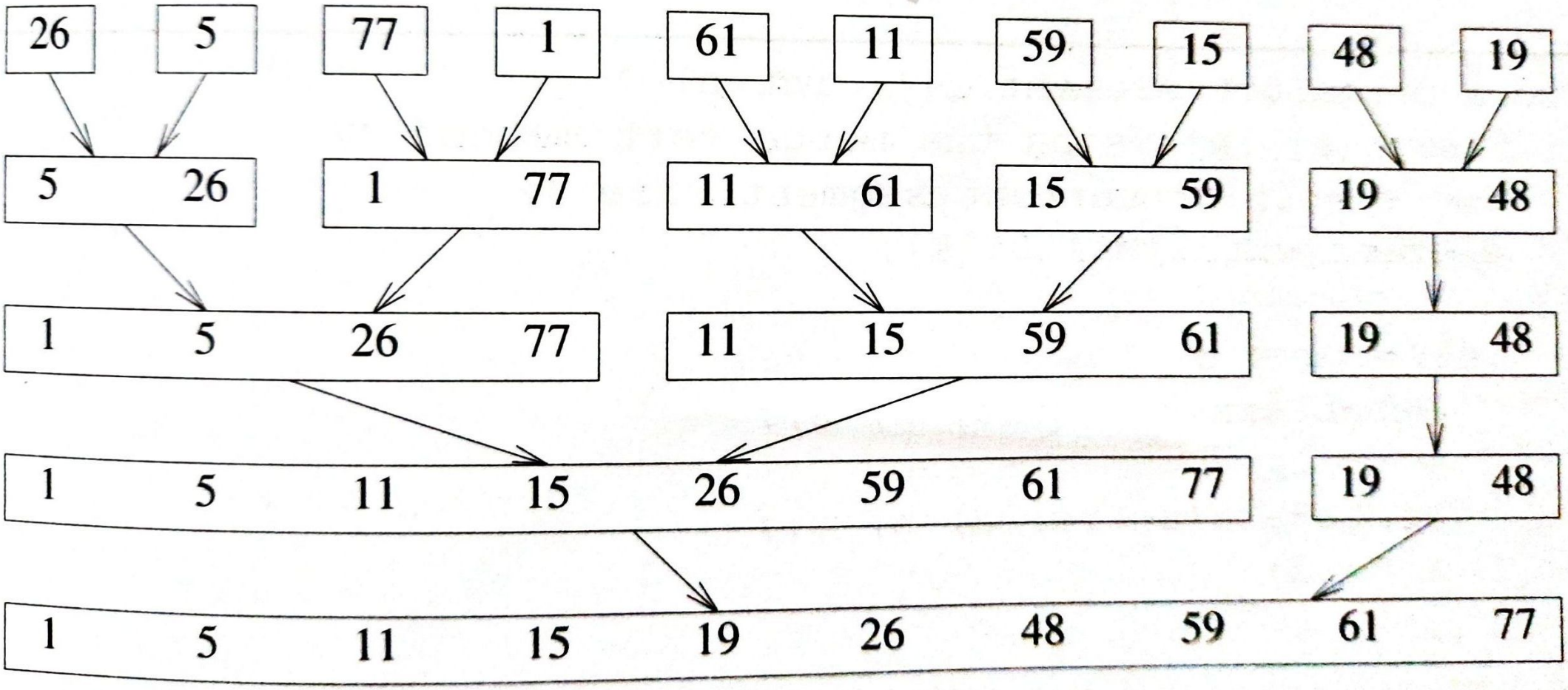

26 5 77 1 61 11 59 15 48 19
5 26 1 77 11 61 15 59 19 48
1 5 26 77 11 15 59 61 19 48
1 5 11 15 26 59 61 77 19 48
1 5 11 15 19 26 48 59 61 77

Figure 7.4: M

```
void merge(element initList[], element mergedList[],
           int i, int m, int n)
{/*  the sorted lists initList[i:m] and initList[m+1:n] are
     merged to obtain the sorted list mergedList[i:n] */
   int j,k,t;
   j = m+1;          /* index for the second sublist */
   k = i;            /* index for the merged list */

   while (i <= m && j <= n) {
      if (initList[i].key <= initList[j].key)
         mergedList[k++] = initList[i++];
      else
         mergedList[k++] = initList[j++];
   }
   if (i > m)
   /* mergedList[k:n] = initList[j:n] */
      for (t = j; t <= n; t++)
         mergedList[t] = initList[t];
      else
      /* mergedList[k:n] = initList[i:m] */
         for (t = i; t <= m; t++)
            mergedList[k+t-i] = initList[t];
}
```