Quiz3 Solutions

1. *Given an array of 100 elements, the Shellsort algorithm requires?*
Given an array of size *n*, the shell sort algorithm starts with an increment of *n*/2 to form lists of size 2, then modify the increment to *n*/4 to generate lists of size 4, and so on down to the smallest increment that is greater than 2 (see code on Page 228)
Therefore, for *n*=100, the increments are 50, 25, 12, 6, and 3. Since the algorithm makes a pass for each increment in addition to a final pass, then **the total number of passes is 6**

2. *The following code snippet*

```
for (int i=1; i<n; i++)
     for (int j=i; j>0; j--)
          if (A[j]<A[j-1])
               swap(A, j, j-1);
```

represents the Insertion sort algorithm (Page 221)

3. *The Bubble Sort algorithm* **makes the same number of comparisons** regardless of the state of the array. For each comparison however, the algorithm may or may not perform a swap (depends on the relative ordering of the two elements). A swap costs time and hence, the time varies depending on the ordering of the data in the array, which also means that **a partially sorted array takes less time to sort than a reverse-sorted array** (less number of swaps).

4. *The Quicksort algorithm* chooses an element as pivot (the idea is that this element is chosen at random, but since calling the random generator function takes time, the middle element is chosen to simulate picking an element randomly). Through the partitioning function, the algorithm places the pivot in its final sorted position within the array. This process continues to cover each element in the array (every single element will be chosen as a pivot during the process and hence, each element will be assigned to its final sorted position). **Therefore, the algorithm picks elements, one at a time, and places them in their final sorted positions.**

5. Within each pass, *the Selection Sort algorithm* remembers the index of the element to sort and at the end of the pass, it makes a single swap. Hence, there is a single swap per pass, which is less than the number of swaps made by the Bubble Sort and Insertion Sort algorithm for any given set of data. Hence, **the Selection Sort algorithm makes the least number of swaps.**

6. **The Shell Sort algorithm call the Insertion Sort algorithm** within each pass and makes a final call to it before ending (see code on Page 228)

7. **Given a permutation of size *n* (array containing numbers between 1 and *n*), the algorithm can sort the array in *n* steps.**

8. Algorithm that is best suited for a linked list:
1) The Shellsort algorithm uses the Insertion sort algorithm, which **walks through the list elements backward**.
2) The Quicksort algorithm **requires random access to list elements** (partitioning the array and accessing the pivot).
3) The Heapsort algorithm also **requires random access to list elements** (to access the root node and perform a siftdown).
4) The Radix Sort **requires random access to list elements** (to position the elements according to the digit value that is being examined).
5) **The Mergesort algorithm does NOT require random access and hence, it is the most suitable algorithm for singley linked lists.**

9. The binary sort algorithm stores records in an array of linked lists. The size of the array is equal to the maximum key value, MaxKeyVal. Identical key values are stored in the bins (nodes) of the same linked list. Hence if the array to be sorted has *n* identical values, the number of bins at the corresponding array position will be ***n* (maximum number of bins)**. If on the other hand, the array to be sorted has no particular key value between 0 and MaxKeyVal-1, the number of bins at the corresponding array position will be **zero (minimum number of bins)**.
Example 1:

Array to be sorted={1,2,4,5,6,0,1,0,1}
MaxKeyVal=10 (upper limit) $\Rightarrow$ BinSort uses an array of 10 linked lists
n=9
Number of bins at array positions 3, 7, 8, 9 is zero (missing key values from the array to be sorted)
Number of bins at array position 1 is 3 (3 1's in the array to be sorted)

Example 2:

Array to be sorted={4,4,4,4,4,4,4,4}
MaxKeyVal=7 $\Rightarrow$ BinSort uses an array of 7 linked lists
n=8
Number of bins at array position 4 is 8=***n*** (identical key values)
Number of bins at array positions 0, 1, 2, 3, 5, 6 is **zero** (missing key values from the array to be sorted)

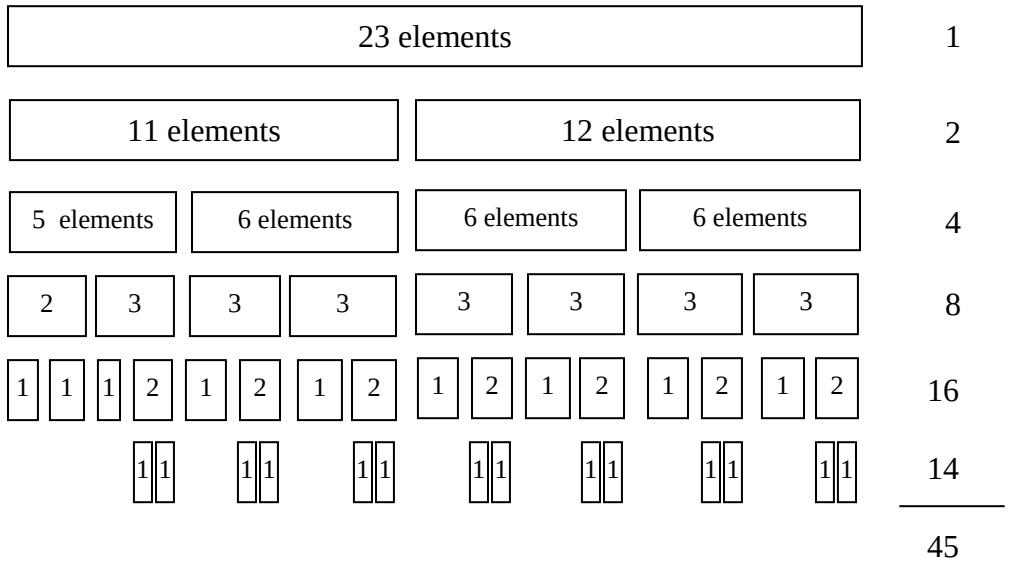
10. To produce a sorted list, **the Heapsort algorithm uses a max-heap**. This is so we place the max value in array position *n*-1, then the next max value in array position *n*-2, and so on until we end up with the minimum value in array position 0.

11. The Radix Sort utilizes temporary arrays of equal size as the original array for sorting.

12. The theory of *the Radix Sort algorithm* uses an array of linked lists. **In practice however, the Radix Sort uses an array of the same size as the original array** (see code and illustration on Pages 244 through 245).

13. *The Radix Sort algorithm* **works as follows**:
1) Inspect the rightmost digit of the records' key values in the array and place them in bins of an array of linked lists accordingly
2) Inspect the next digit of the key values in the bins (going through the bins left to right, top to bottom) and place them in different bins accordingly
3) Repeat step 2) until the leftmost digit of the key values is inspected
4) Output the records according to the key values in the bins by going through the bins left to right, top to bottom

14. Every time the Mergesort algorithm is called, it divides the original array into two halves and calls itself recursively for every half until a size of one element is reached. Take the following example:

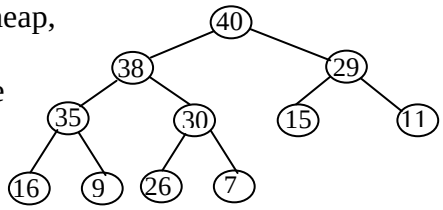


$$\left\lfloor \left\lceil \frac{23}{1} \right\rceil \right\rfloor + \left\lfloor \left\lceil \frac{23}{2} \right\rceil \right\rfloor + \left\lfloor \left\lceil \frac{23}{4} \right\rceil \right\rfloor + \left\lfloor \left\lceil \frac{23}{8} \right\rceil \right\rfloor + \left\lfloor \left\lceil \frac{23}{16} \right\rceil \right\rfloor = 23 + 12 + 6 + 3 + 1 = 45$$

15. *After picking the pivot, placing it at the end of the array, partitioning the array, and then repositioning the pivot*; **the pivot could end up anywhere in the array**. The new position becomes the final position of the pivot in the sorted array and hence, **all elements to its left are smaller than its value, while all elements to its right are greater (or equal) to its value.**

16. When values are introduced one by one to a max heap, nodes are added top to bottom – left to right.
Hence, for the maxheap on the right, values must have been introduced in the following sequence:

40, 38, 29, 35, 30, 15, 11, 16, 9, 26, 7



17. If we delete the node with the minimum key value and then immediately insert the same value back, none of the binary tree structures will remain unchanged in all cases.

a. Binary Search Tree: if the node with minimum key has a right child, then we will end up with a different tree.
b. Min-Heap: The same node (with minimum key value) will end up at the root again, but the internal structure of the tree may change.
c. Max-Heap: The node with minimum key value may not be the right-most node in the lowest level, and hence, if it is removed and then inserted back, its position will change.