Sorting

Elementary Sorts

1. What is the maximum number of exchanges involving any particular item during selection sort? What is the average number of exchanges involving one specific item x?
   Solution. The average number of exchanges is exactly 2 because there are exactly n exchanges and n items (and each exchange involves two items). The maximum number of exchanges is n.

2. Which method runs fastest for an array with all keys identical, selection sort or insertion sort?
   Solution. Insertion sort runs in linear time when all keys are equal.

3. Suppose that we use insertion sort on a randomly ordered array where items have only one of three key values. Is the running time linear, quadratic, or something in between?
   Solution. Quadratic. Insertion performon me mire kur vektori eshte gjysme i renditur dhe jo random.

4. Why not use selection sort for h-sorting in shellsort?
   Solution. Insertion sort is faster on inputs that are partially-sorted. Selection sort i ka best, average dhe worst case te gjitha kompleksitet ne katror.

5. **To do**. Minimum number of moves to sort an array. Given a list of N keys, a move operation consists of removing any one key from the list and appending it to the end of the list. No other operations are permitted. Design an algorithm that sorts a given list using the minimum number of moves.

6. Binary insertion sort. Develop an implementation BinaryInsertion.java of insertion sort that uses binary search to find the insertion point j for entry a[i] and then shifts all of the entries a[j] to a[i-1] over one position to the right. The number of compares to sort an array of length n should be ~ n lg n in the worst case. Note that the number of array accesses will still be quadratic in the worst case.

7. Insertion sort without exchanges. Develop an implementation InsertionX.java of insertion sort that moves larger items to the right one position rather than doing full exchanges.

Mergesort, Quicksort

1. Does the abstract inplace merge produce proper output if and only if the two input subarrays are in sorted order? Prove your answer, or provide a counterexample.
   Solution. Yes. If the subarrays are in sorted order, then the inplace merge produces proper output. If one subarray is not in sorted order, then its entries will appear in the output in the same order that they appear in the input (with entries from the other subarray intermixed).

2. Suppose that top-down mergesort is modified to skip the call on merge() whenever a[mid] <= a[mid+1]. Prove that the number of compares used for an array in sorted order is linear.
   Solution. Since the array is already sorted, there will be no calls to merge(). When N is a power of 2, the number of compares will satisfy the recurrence T(N) = 2 T(N/2) + 1, with T(1) = 0.

3. Faster merge. Implement a version of merge() that copies the second half of a[] to aux[] in decreasing order and then does the merge back to a[]. This change allows you to remove the code to test that each of the halves has been exhausted from the inner loop. Note: the resulting sort is not stable.

```
private static void merge(Comparable[] a, int lo, int mid, int hi) {
   for (int i = lo; i <= mid; i++)
      aux[i] = a[i];

   for (int j = mid+1; j <= hi; j++)
      aux[j] = a[hi-j+mid+1];

   int i = lo, j = hi;
   for (int k = lo; k <= hi; k++)
      if (less(aux[j], aux[i])) a[k] = aux[j--];
      else                      a[k] = aux[i++];
}
```

4. Inversions. Develop and implement a linearithmic algorithm Inversions.java for computing the number of inversions in a given array (the number of exchanges that would be performed by insertion sort for that array—see Section 2.1). This quantity is related to the Kendall tau distance; see Section 2.5.

5. Merging two arrays of different lengths. Given two sorted arrays a[] and b[] of sizes M and N where $M \geq N$, devise an algorithm to merge them into a new sorted array c[] using ~ N lg M compares.
   Hint: use binary search.

Note: there is a lower bound of Omega(N log (1 + M/N)) compares. This follows because there are M+N choose N possible merged outcomes. A decision tree argument shows that this requires at least lg (M+N choose N) compares. We note that n choose r >= (n/r)^r.

6. Indirect sort. Develop and implement a version of mergesort that does not rearrange the array, but returns an int[] array perm such that perm[i] is the index of the i th smallest entry in the array.

7. About how many compares will Quick.sort() make when sorting an array of N items that are all equal?
   Solution. ~ N lg N compares. Each partition will divide the array in half, plus or minus one.

Priority Queues, Heapsort

1. Is an array that is sorted in decreasing order a max-oriented heap.
   Answer. Yes.

2. Suppose that your application will have a huge number of insert operations, but only a few remove the maximum operations. Which priority-queue implementation do you think would be most effective: heap, unordered array, ordered array?
   Answer. Unordered array. Insert is constant time.

3. Design a linear-time certification algorithm to check whether an array pq[] is a min-oriented heap.
   Solution. See the isMinHeap() method in MinPQ.java.

4. Add a min() method to MaxPQ.java. Your implementation should use constant time and constant extra space.
   Solution: add an extra instance variable that points to the minimum item. Update it after each call to insert(). Reset it to null if the priority queue becomes empty.

5. Given a maximum oriented binary heap, design an algorithm to determine whether the kth largest item is greater than or equal to x. Your algorithm should run in time proportional to k.
   Solution: if the key in the node is greater than or equal to x, recursively search both the left subtree and the right subtree. Stop when the number of node explored is equal to k (the answer is yes) or there are no more nodes to explore (no).

6. Dynamic-median finding. Design a data type that supports insert in logarithmic time, find the median in constant time, and remove the median in logarithmic time.

Solution: We use two heaps: Max and Min heap. At max heap we store the smaller half of the elements. The root represents the largest of the lower halfs. At min we store the larger half of the elements. The root represents the smallest element of the upper half.