Problem 5-1 (Comparisons) 8 points

Consider a sorted array $A$ of $n$ elements and two integers $x$ and $y$ not in the array, with $x \leq y$. A comparison based algorithm computes how many elements in $A$ are less than both $x$ and $y$, how many elements are between $x$ and $y$ and how many are bigger than both $x$ and $y$. What is the best lower bound (precise answer, not asymptotic) you can prove (using the decision tree technique) for the time complexity of the algorithm?

Problem 5-2 (Nuts and Bolts) 15 points

Assume that we are given $n$ bolts and $n$ nuts of different sizes, where each bolt exactly matches one nut. Our goal is to find the matching nut for each bolt. The nuts and bolts are too similar to compare directly; however, we can test whether any nut is too big, too small, or the same size as any bolt.

(a) (4 points) Prove that in the worst case, $\Omega(n \log n)$ nut-bolt tests are required to correctly match up the nuts and bolts.

(b) (6 points) Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find $k$ arbitrary matching pairs. (Hint: prove two separate lower bounds: $\Omega(n)$ and $\Omega(k \log n)$.)

(c) (5 points) Give a randomized algorithm that runs in expected time $O(n)$ and finds the $k$-th largest nut given any integer $k$. You may assume that it is possible to efficiently sample a random nut/bolt.

Problem 5-3 (Faster Counting Sort) 12 points

We will consider a variant of the counting sort algorithm that sorts an $n$ element array $A$ whose elements are integers between 1 and $k$.

(a) (4 pts) Recall the code of standard counting sort.

```python
COUNTING_SORT(A, B, k,)
1 let C be a new array
2 For $i = 1$ to $k$
3     $C[i] = 0$
4 For $j = 1$ to $n$
5     $C[A[j]] = C[A[j]] + 1$
```

PS5 Page 1
For \( i = 1 \) to \( k \)
\[ C[i] = C[i] + C[i - 1] \]
For \( j = n \) to \( 1 \)
\[ C[A[j]] = C[A[j]] - 1 \]
Return \( B \)

After the run of the counting sort, somebody accidentally erased both array \( A \) and \( B \), but only left array \( C \). Design an \( O(n) \) procedure \textsc{recoverBfromC} that still reconstructs the sorted array \( B \) from \( C \). Why does it only work when sorting integers?

(b) (4 pts) Now, consider the following variant of the Counting Sort algorithm that sorts the \( n \)-element array \( A \) whose elements are integers between 1 and \( k \).

```python
CountingSortFast(A, k, n)
For i = 1 to k
\[ C[i] = 0 \]
For j = 1 to n
\[ C[A[j]] = C[A[j]] + 1 \]
\[ b = 1 \]
For i = 1 to k
\[ For j = 1 to C[i] \]
\[ A[b] = i \]
\[ b = b + 1 \]
Return A
```

Formally argue the correctness of \textsc{CountingSortFast}.

(c) (4 pts) Notice that while \textsc{CountingSort} makes in time \( 3n + 2k \) array assignments, but \textsc{CountingSortFast} makes only in time \( 2n + k \) array assignments. Further \textsc{CountingSortFast} does not use an extra array \( B \) used by \textsc{CountingSort}. Explain what the problem with \textsc{CountingSortFast}. In which realistic use cases would one prefer slightly slower \textsc{CountingSort}? Justify your answer.

Problem 5-4 (Choosing the Right Tool) 9 points

For each example choose one of the following sorting algorithms and carefully justify your choice: \textsc{HeapSort}, \textsc{RadixSort}, \textsc{CountingSort}. Give the expected runtime for your choice as precisely as possible. If you choose \textsc{Radix Sort} then give a concrete choice for the basis (i.e. the value of “\( r \)” in the book) and justify it. (\textbf{Hint}: We assume that the array itself is stored in memory, so before choosing the fastest algorithm, make sure you have the space to run it!)

(a) Sort the length \( 2^{16} \) array \( A \) of 128-bit integers on a device with 100MB of RAM.
(b) Sort the length \( 2^{24} \) array \( A \) of 256-bit integers on a device with 600MB of RAM.
(c) Sort the length \( 2^{16} \) array \( A \) of 16-bit integers on a device with 1GB of RAM.
Problem 5-5 (Finding the Major Elements) 15 points

Let us say that a number $x$ is $c$-major for an $n$-element array $A$, if more than $n/c$ elements of $A$ are equal to $x$.

(a) (6 pts) Give $O(n)$-time algorithm to find all 2-major elements of $A$. How many could there be?

(b) (9 pts) Give $O(cn)$-time algorithm to find all $c$-major elements of $A$. How many could there be?