**Question 1** (Sorting Runtime, 30 points). Consider the following sorting algorithm:

```plaintext
RecursiveSort(A, i, j) \ sorts the elements of A between indices i and j
  if j = i + 1
    if A[i] > A[j]
      swap(A[i], A[j])
  else
    Set k = ceiling(2*(j-i+1)/3)-1
    RecursiveSort(A, i, i+k)
    RecursiveSort(A, j-k, j)
    RecursiveSort(A, i, i+k)
```

(a) Give a recurrence for the runtime of this algorithm on an array of length n (when i = 1 and j = n). [10 points]

If $T(n)$ is the runtime when $j - i + 1 = n$, we have that

$$
T(n) = \begin{cases} 
O(1), & \text{if } n = O(1) \\
3T(2n/3 + O(1)) + O(1), & \text{otherwise.}
\end{cases}
$$

(b) What is the asymptotic runtime of this algorithm? [20 points]

We apply the Master Theorem with $a = 3$, $b = 3/2$, $d = 0$. Clearly $a > b^d$ so the asymptotic runtime is $\Theta(n^{\log_b a}) = \Theta(n^{\log_{3/2} 3}) = \Theta(n^{2.7...})$. 

---

1
**Question 2** (Escape, 35 points). Ashley is trying to escape from a burning building. She has an undirected graph $G$ mapping out the building, with edge weights denoting how long it will take her to traverse each edge. Her current location is a vertex $s$ and the exit is at a vertex $t$. Unfortunately, as the building burns, some of the vertices may become inaccessible. Each vertex $v$ has a time $T_v$ after which it will become impassible.

Give an algorithm that given $G, s, t$ the weights and the times $T_v$ determines whether or not it is possible for Ashley to escape. For full credit, your algorithm should run in time $O(|V| \log(|V|) + |E|)$ or better.

We modify Dijkstra's algorithm so that we only update the distance to a vertex $w$ if the new distance would be less than $T_w$ (since Ashley cannot make it to that vertex in that case). We then check if the distance to $t$ is less than infinity. In particular the algorithm is:

Let Q be a PQ (implemented with a Fib. Heap)
For v in V
  d(v) <- infinity
  If v=s
    d(v) <- 0
    Q.Insert(v)
While(Q not empty)
  v <- Q.DeleteMin()
  For (v,w) in E
    dnew <- d(v) + length(v,w)
    If dnew <= min(d(w),T_w)
      d(w) <- dnew
      Q.DecreaseKey(w)
  If d(t) < infinity
    Return ‘Escape Possible’
  Else
    Return ‘Escape Impossible’

The runtime analysis is the same as the usual Dijkstra's algorithm with a Fibonacci Heap, and thus the final runtime is $O(|V| \log(|V|) + |E|)$.

As for the proof of correctness, it is easy to see that for $w$ in the priority queue that $d(w)$ is always the minimum value of $d(v) + \ell(v,w)$ for $v$ adjacent to $w$ and removed from the queue if this is less than $T_w$ and infinity otherwise. We claim by induction that whenever a vertex $v$ is removed from the priority queue that $d(v)$ is the earliest that Ashley can safely reach $v$ and infinity if she cannot safely reach $v$.

The base case is $s$ (the first vertex removed), for which it is clear that the earliest she can reach it is at time 0.

Assuming that all removed vertices have the correct time, in order to reach any non-removed vertex, Ashley would first need to reach some removed vertex $v$ (in time at least $d(v)$) and then traverse an edge to some non-removed vertex $w$. She would arrive at $w$ at time $d(v) + \ell(v,w)$ and this would only be safe if this is less than $T_w$. If you take the minimum value of this over all non-removed $w$’s (which will give the next vertex to be removed from the priority queue), this will be the earliest time that Ashley can safely reach any non-removed vertex, and thus will be the earliest time she can safely reach $w$. This completes our inductive step.

From this, we have that Ashley can safely reach $t$ if and only if $d(t) < \infty$, which is what our algorithm checks.
Question 3 (Finding a Local Maximum, 35 points). Let \(A\) be an unsorted array of \(n\) real numbers, \(A[1], A[2], \ldots, A[n]\). We say that \(A\) has a local maximum at \(k\) if \(1 < k < n\) and \(A[k] \geq \max(A[k-1], A[k+1])\) or \(k = 1\) and \(A[1] \geq A[2]\) or if \(k = n\) and \(A[n] \geq A[n-1]\). Give an algorithm that given the array \(A\) finds a local maximum.

For full credit, your algorithm should run in time \(O(\log(n))\).

We define a procedure \(\text{LocalMax}(i, j)\) which finds a local maximum \(i \leq k \leq j\) if \(i\) is either 1 or \(A[i] \geq A[i-1]\) and \(j\) is either \(n\) or \(A[j] \geq A[j+1]\). The algorithm is as follows:

\[
\text{LocalMax}(i, j) \\
\quad \text{If } j = i \\
\quad \quad \text{Return } i \\
\quad \text{If } j = i+1 \\
\quad \quad \text{If } A[i] > A[j] \\
\quad \quad \quad \text{Return } i \\
\quad \quad \text{Else} \\
\quad \quad \quad \text{Return } j \\
\quad \text{Let } k = \text{floor}((i+j)/2) \\
\quad \text{If } A[k] > A[k+1] \\
\quad \quad \text{Return } \text{LocalMax}(i, k) \\
\quad \text{Else} \\
\quad \quad \text{Return } \text{LocalMax}(k+1, j)
\]

To analyze the runtime, we note that this algorithm does \(O(1)\) work and then makes a recursive call on a sublist of roughly half the size. Thus, by the Master Theorem, the final runtime is \(O(\log(n))\).

For correctness, we proceed by induction on \(j-i\). We note that if \(j = i\) then by assumption \(A[i] \geq A[i-1]\) and \(A[i] \geq A[i+1]\) and so \(i\) is a local maximum. If \(j = i+1\) and \(A[j] \geq A[i]\) then since \(A[j] \geq A[j+1]\), \(j\) is a local maximum and the other case follows similarly. When \(j-i \geq 2\), if \(A[k+1] \geq A[k]\) then the assumptions necessary for \(\text{LocalMax}(k+1, j)\) to return a local maximum are satisfied so by the inductive hypothesis, the output is correct. Things follow similarly when \(A[k] > A[k+1]\).

This completes our proof.