

CSCI303 Fall 2006 Homework 3 Solutions

9-1) last updated fall 2006, Yuriy Brun, ybrun@usc.edu

a) Sort the n numbers using Mergesort which take $O(n \lg n)$ time. Put the i largest elements from the sorted array into the output array, in $O(i)$ time. Therefore total worst case time is $O(n \lg n) + O(i) = O(n \lg n)$.

b) Build a Max-Heap, which takes $O(n)$ time, and call Extract-Max i times, which takes $O(\lg n)$ time per call. Thus overall the procedure takes $O(n + i \lg n)$ time.

c) To find the i_{th} largest number, use the Select algorithm, which takes $O(n)$ time. Partition the array of n numbers with this number, using the Partition algorithm, which takes $O(n)$ time. Finally, sort the i numbers using Mergesort in $O(i \lg i)$ time. So total time is $O(n + i \lg i)$.

9.2-4) last updated fall 2006, Yuriy Brun, ybrun@usc.edu

Worst case performance of RANDOMIZED-SELECT will result due to the worst case performance of RANDOMIZED-PARTITION. Given that we have to find the minimum element, the worst case performance of RANDOMIZED-PARTITION will result if each call to it results in a partitioning around the maximum element left so far (i.e the maximum element is always chosen as the pivot).

3	2	9	0	7	5	4	8	6	1
---	---	---	---	---	---	---	---	---	---

Partition with pivot = 9

3	2	1	0	7	5	4	8	6
---	---	---	---	---	---	---	---	---

pivot = 8

3	2	1	0	7	5	4	6
---	---	---	---	---	---	---	---

pivot = 7

3	2	1	0	6	5	4
---	---	---	---	---	---	---

pivot = 6

3	2	1	0	4	5
---	---	---	---	---	---

pivot = 5

3	2	1	0	4
---	---	---	---	---

pivot = 4

3	2	1	0
---	---	---	---

pivot = 3

2	1	0
---	---	---

pivot = 2

1	0
---	---

pivot = 1

0

The worst case running time of RANDOMIZED-SELECT is $\Theta(n^2)$, also obtainable by solving the following recurrence: $T_{RS}^{WC}(n) = T_{RS}^{WC}(n-1) + \Theta(n)$.

9.3-1) last updated fall 2006, Yuriy Brun, ybrun@usc.edu

For groups of 7, the algorithm still works in linear time. The number of elements greater than x (similarly number less than x) is at least $4\left(\frac{1}{2}\left\lceil\frac{n}{7}\right\rceil - 2\right) \geq \frac{2n}{7} - 8$, so the recurrence would be

$T(n) \leq T\left(\frac{n}{7}\right) + T\left(\frac{5n}{7} + 8\right) + \Theta(n)$, which satisfies a linear function (can be shown by following the groups-of-5 proof or by substitution method).

For groups of 3, this algorithm no longer works in linear time. The recurrence becomes $T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3} + 3\right) + \Theta(n)$. Using substitution method we can show that for no choice

of c , will $T(n) \leq cn$. Thus $T(n)$ does not have a linear solution. Intuitively, we recurse on $\frac{2n}{3}$ and $\frac{n}{3}$, so we have no part of n to help cancel out the positive terms (for the groups-of-5 case we

recursed on $\frac{n}{5}$ and $\frac{7n}{10}$ giving us a total of $\frac{9n}{10}$ and some room to work with $\left(-\frac{1n}{10}\right)$ to cancel the positive terms).

9.3-5) last updated fall 2006, Yuriy Brun, ybrun@usc.edu

```
Simple-Select(A, p, q, i) {
    Swap(A[q], A[index_of_median(A));
    Partition(A);
    if  $\left(i = \left\lfloor\frac{n}{2}\right\rfloor\right)$  return A[i];
    else if  $\left(i < \left\lfloor\frac{n}{2}\right\rfloor\right)$  return Simple-Select(A, p,  $\left\lfloor\frac{n}{2}\right\rfloor - 1, i$ );
```

```

else if  $\left(i > \left\lfloor \frac{n}{2} \right\rfloor\right)$  return Simple-Select $\left(A, \left\lfloor \frac{n}{2} \right\rfloor + 1, q, i - \left\lfloor \frac{n}{2} \right\rfloor - 1\right)$ ;
}

```

$T_{SS}^{WC}(n) = T_{SS}^{WC}\left(\frac{n}{2}\right) + \Theta(n)$. By the master method, case 3: $T_{SS}^{WC}(n) = \Theta(n)$.

9.3-8) last updated fall 2006, Yuriy Brun, ybrun@usc.edu

```

Median2n(X, x_first, x_last, Y, y_first, y_last) {
    if x_first > x_last or y_first > y_last
        return X(x1)
    x_med ← (x_first + x_last)/2
    y_med ← (y_first + y_last)/2
    if X(x_med) = Y(y_med)
        return X(x_med)
    else if X(x_med) < Y(y_med)
        return Median2n(X, x_med + 1, x_last, Y, y_first, y_med - 1)
    else
        return Median2n(X, x_first, x_med - 1, Y, y_med + 1, y_last)
}

```

To find the median, call Median2n(X, 1, n, Y, 1, n).

First, the algorithm checks if the medians of the two lists are equal. If they are, then that has to be the median of the union of the two lists. If they are not, then we can illuminate the $\left\lfloor \frac{n}{2} \right\rfloor$ elements larger or equals than the larger of the two medians, and the $\left\lceil \frac{n}{2} \right\rceil$ elements smaller or equal than the smaller of the two medians and recurse on the elements left to find their median, which will be the median of the original union.

The running time for this algorithm is $T_{M2}^{WC}(n) \leq T_{M2}^{WC}\left(\frac{n}{2}\right) + \Theta(1)$, so $T_{M2}^{WC}(n) = O(\lg n)$ by the 1st case of the Master Theorem Prime.