Morgan Baccus
Cpts 350
Homework #5

Problem #1

* Iterate through A from left to right while saving the max and min values in variables called max and min respectively.

* While iterating through A, we will read in two values and call them var1 and var2.

* We can only do 3 comparisons as $1.5 * n$, where $n=2$, $= 3$.
* The 3 comparisons are:

  1. Compare var1 to var2 (without loss of generality) and assume var1 is larger than var2.

  2. Compare var1 to max, the larger value is now set to max.

  3. Compare var2 to min, the smaller value is now set to min.

Morgan Baccus

Problem #2

Algorithm: S(A, n, i):

* Copy elements of A to new array called B

* Create a new array called C with size i used to store the results

* Start loop at j=0 to i-1
  - Compute min value of B
  - Store min value in C
  - remove min value from B

* Return C

Run time of S(A, n, i):

* Outer loop runs for i number of iterations
* Inner loop runs n times at most to find the min
* All other operations are constant
* Total run time = $O(i \times n) + C = O(i \times n)$

Algorithm: T(A, n, i):

* Copy elements of A to a new array called B

* Create a new array called C with size i used to store the results

* Use mergesort to sort B from sallest to largest
* Start loop at j=0 to i-1
  - Store value of A[i] in C[j]

* return C

Morgan Baccus

Problem #2 Cont.

Run time of $T(A, n, i)$:

* Merge sort takes $O(n \log n)$ to sort an array of size n
* Loop to create C will take i number of iterations
* Other operations are constant
* Total run time = $O(n \log n) + O(i) + C = O(n \log n)$

When S performs better than T:

S performs better than T when $i > \log n$.

Example: $i = 2$, $n = 16$

$S(A, 2, 16) = O(i \times n) = O(2 \times 16) = O(32)$

$T(A, 2, 16) = O(n \log n) = O(16 \log 16) = O(64)$

When T performs better than S:

T performs better than S when $i > \log n$.

Example: $i = 12$, $n = 16$

$S(A, 12, 16) = O(i \times n) = O(12 \times 16) = O(192)$

$T(A, 12, 16) = O(n \log n) = O(16 \log 16) = O(64)$

Morgan Baccus

Problem #3

K = 3

Total number of medians: $n/3$

Elements less than the median of medians: $2 \times 1/2 \times n/3 = n/3$

Elements greater than the median of medians: $2 \times 1/2 \times n/3 = n/3$

So, worst case location of median of medians: $n - n/3 = 2n/3$

Hence, the worst case complexity for k = 3:

$$T_W(n) \leq \Theta(n) + T_W(n/3) + T_W(2n/3)$$

k = 7

Total number of medians: $n/7$

Elements less than the median of medians: $4 \times 1/2 \times n/7 = 2n/7$

Elements greater than the median of medians: $4 \times 1/2 \times n/7 = 2n/7$

So, worst case location of median of medians: $n - 2n/7 = 5n/7$

Hence, the worst case complexity for k = 7:

$$T_W(n) \leq \Theta(n) + T_W(n/7) + T_W(5n/7)$$

Morgan Baccus

Problem #4

Worst case:

The wost case complexity of the algorith is the max of the following, given r:

* Worst case of quickselect on the low part: $O(r^2)$
* Worst case of the linear select on the high part: $O(n-r)$

∴ the worst case complexity is $O(\max_{1 \le r \le n}(r^2, n-r))$ which is $O(n^2)$ for $r=n$ given any r.

Average Case:

Average case is denoted by $T_{avg}(n)$. We do not include i here because we are using random input and i is also random.

A will be partioned into a high and low part where the high part has n-r elements and the low part has r-1 elements.

The input is random, so r can be at any position in A between 1 and n. There is equal probability r will be at any index.

Since i is also random, it can be at any index as well. The probability that i is the same as $r = \frac{1}{n}$, i is in the low part $= \frac{r-1}{n}$, and i is in the high part $= \frac{n-r}{n}$.

Morgan Baccus

Problem #4 cont.

$$T_{avg}(n) = \Theta(n) + \frac{1}{n} \sum_{1 \le r \le n} \left(O(1) \times \frac{1}{n} + T_{qs.avg}(r-1) \times \frac{r-1}{n} + T_{is.avg}(n-r)\frac{n-r}{n} \right)$$

$$T_{avg}(n) = \Theta(n) + \frac{1}{n} \sum_{1 \le r \le n} \left(O(1) \times \frac{1}{n} + O(r-1) \times \frac{r-1}{n} + O(n-r) \times \frac{n-r}{n} \right)$$

Simplify...

avg. case:

$$\boxed{T_{avg}(n) = \Theta(n) + \frac{2}{n} \sum_{1 \le r \le n} O(r-1) \times \frac{r-1}{n}}$$

Morgan Baccus

Problem #9

* Create array N with n number of distinct elements. Here, $n=5$.

* Loop through N n number of times. (5)
    * Compare N[0] with N[n]
    * If N[n] < N[0], switch them
    * If not, increment by one

Hence, the minimum number of operations is =

$$\boxed{\frac{n(n-1)}{2}}$$