Cpt S 450 Homework #2 Solutions

1. (easy) Write psuedo-code for partition(A, p, q).

```
int partition(A, p, q){
    //make sure that the parameters are valid
  assert(p ≤ q);
  if p == q return p;
    //in below, p < q
    //put the first element in a garbage can
  garbagecan=A[p];
    //set mode (which is either I or J) to be I;
  mode=I;
    //assign two cursors i = p and j = q
    //where i moves to the right and j moves to the left
    //when they meet, we are done
  i = p; j = q;
  while i < j{
   if mode==I,
     if A[j] ≤ garbagecan,
      we swap A[j] and A[i], and set mode to be J;
     else //A[j] > garbagecan
      j − −;
   if mode==J,
     if A[i] > garbagecan,
      we swap A[j] and A[i], and set mode to be I;
     else //A[i] ≤ garbagecan
      i + +;
  }//while
    //reclaim the garbage
  A[i] =garbagecan;
    //now, the i is the r
  return i;
}//partition
```

2. (standard) Consider insertsort. Suppose that the input array A has 1% probability to be monotonically decreasing. Show that, in this case, the average-case complexity of insertsort is $\Theta(n^2)$.

We use T_{avg1} to denote the demanded average-case complexity. We also use T_W to denote the worst case (when the input is monotonically decreasing) complexity of insertsort, which is known to be $\Theta(n^2)$. Finally, we use T_{avg} to denote the average-case complexity of insertsort.

Now, (why?)

$T_{avg1}(n) = \frac{1}{100} \cdot T_W(n) + \frac{99}{100} \cdot T_{avg}(n)$.

Notice that $T_W(n) = \Theta(n^2)$ and $T_{avg}(n) = O(n^2)$. This gives,

$\Theta(n^2) + 0 \leq T_{avg1}(n) \leq \Theta(n^2) + O(n^2) = \Theta(n^2)$.

So, $T_{avg1}(n) = \Theta(n^2)$.

3. (not hard) Let iqsort$(A, 1, n)$ be an algorithm that sorts an array A with n integers. It works as follows:

```
iqsort(A, p, q){
if p ≥ q, return;
r=partition(A, p, q);
//run quick sort on the low part
quicksort(A, p, r − 1);
//run insert sort on the high part
insertsort(A, r + 1, q);
}
```

Compute the best-case, worst-case, and average-case complexities of iqsort.

We use $T(n)$ to denote a particular run of iqsort over some A with n elements. Clearly, $T(n)$ is the summation of

- $\Theta(n)$ – the cost of partition
- $T^{\text{quicksort}}(r-1)$ – the cost of quicksort over the low-part
- $T^{\text{insertsort}}(n-r)$ – the cost of insertsort over the high-part

That is,

$$T(n) = \Theta(n) + T^{\text{quicksort}}(r-1) + T^{\text{insertsort}}(n-r).$$

Obviously, to make the $T(n)$ best (smallest), we need consider the cases when both $T^{\text{quicksort}}(r-1)$ and $T^{\text{insertsort}}(n-r)$ are the best. We know that the best case of the quicksort is $T^{\text{quicksort}}(r-1) = \Theta((r-1)\log(r-1))$ and the best of $T^{\text{insertsort}}(n-r) = \Theta(n-r)$. Notice that the best case of the quicksort can not compete with the best case of the insertsort; therefore, the best case of $T(n)$ should be the best case of $T^{\text{insertsort}}(n-r)$ (when the high-part is already sorted) when the low-part becomes empty ($r = 1$); i.e., the best case of $T(n)$ is the best case of $T^{\text{insertsort}}(n-1)$ plus the $\Theta(n)$ (the cost of partition). Hence, the best case of iqsort is still $\Theta(n)$, which is achieved when the input array is already sorted.

Now we consider the formula

$$T(n) = \Theta(n) + T^{\text{quicksort}}(r-1) + T^{\text{insertsort}}(n-r)$$

again. In order to make the $T(n)$ worst (largest), we need consider the cases when both $T^{\text{quicksort}}(r-1)$ and $T^{\text{insertsort}}(n-r)$ are the worst. We know that the worst case of the quicksort is $T^{\text{quicksort}}(r-1) = \Theta((r-1)^2)$ and the worst of $T^{\text{insertsort}}(n-r) = \Theta((n-r)^2)$, which are at the same order. Hence, the worst case of $T(n)$ is achieved when either the low-part or the high-part becomes empty (i.e., one of $r-1$ and $n-r$ becomes 0; that is, one of $T^{\text{quicksort}}(r-1)$ and $T^{\text{insertsort}}(n-r)$ reaches the maximum $\Theta((n-1)^2)$). Hence, the worst case of iqsort is $\Theta((n-1)^2)+\Theta(n) = \Theta(n^2)$. This is achieved when, for instance, either the array is strictly decreasing, or the array starts with the minimal element followed by a strictly decreasing subarray.

Now we consider the formula

$$T(n) = \Theta(n) + T^{\text{quicksort}}(r-1) + T^{\text{insertsort}}(n-r)$$

again. This formula works for a fixed r. On avaerage, we would expect the r appears in every position equally likely – and the low-part and the high-part themselves are random. We use T_{avg} to denote the average case complexity of iqsort. That is,

$$T_{avg}(n) = \sum_{1 \le r \le n} \frac{1}{n}(\Theta(n) + T^{\text{quicksort}}_{avg}(r-1) + T^{\text{insertsort}}_{avg}(n-r)).$$

Recalling the average case of quicksort ($O(n \log n)$, i.e., $\le a \cdot n \log n$ for some a) and the average case of insertsort ($O(n^2)$, i.e., $\le b \cdot n^2$ for some b), we have,

$$T_{avg}(n) \le \Theta(n) + \frac{1}{n} \sum_{1 \le r \le n} a \cdot (r-1)\log(r-1) + b \cdot (n-r)^2.$$

From above, one can show

$$T_{avg}(n) \leq \Theta(n) + \frac{1}{n} \sum_{1 \leq r \leq n} a \cdot n \log n + b \cdot n^2.$$

Therefore,

$$T_{avg}(n) \leq \Theta(n) + \frac{1}{n}\Theta(n^3).$$

We have, $T_{avg}(n) = O(n^2)$.

4. (hard) Let mixsort$(A, 1, n)$ be an algorithm that sorts an array A with n integers. It works as follows:

```
mixsort(A, p, q){
if p ≥ q, return;
r=partition(A, p, q);
//run mixsort on the low part
mixsort(A, p, r − 1);
//run insert sort on the high part
insertsort(A, r + 1, q);
}
```

Compute the best-case, worst-case, and average-case complexities of mixsort.

We use $T(n)$ to denote a particular run of mixsort over some A with n elements. Clearly, $T(n)$ is the summation of

- $\Theta(n)$ – the cost of partition
- $T(r-1)$ – the cost of mixsort over the low-part
- $T^{\text{insertsort}}(n-r)$ – the cost of insertsort over the high-part

That is,

$$T(n) = \Theta(n) + T(r-1) + T^{\text{insertsort}}(n-r).$$

Obviously, to make the $T(n)$ best (smallest), we need consider the cases when both $T(r-1)$ and $T^{\text{insertsort}}(n-r)$ are the best. We know that the best case of the insertsort is $T^{\text{insertsort}}(n-r) = \Theta(n-r)$, which is linear, and the best of $T(n)$ (mixsort) is at least linear. Therefore, the best case of $T(n)$ is when the $r = 1$ (the low-part is empty) and the high-part is the best case of insertsort (the high-part is already sorted). Therefore, the best case of $T(n)$ runs in time $\Theta(n) + \Theta(n-1) = \Theta(n)$, when, for instance, the input array is already sorted.

Consider the formula

$$T(n) = \Theta(n) + T(r-1) + T^{\text{insertsort}}(n-r)$$

again. We use $T_W(n)$ to denote the worst case of mixsort. That is,

$$T_W(n) \le \Theta(n) + T_W(r-1) + T_W^{\text{insertsort}}(n-r).$$

Recall that the worst case $T_W^{\text{insertsort}}(n-r)$ of insertsort is $\Theta((n-r)^2)$. Hence, we have,

$$T_W(n) \le \Theta(n) + T_W(r-1) + \Theta((n-r)^2).$$

By replacing the Θ with constants expressed, we have, for some constants a and b, we have

$$T_W(n) \le a \cdot n + T_W(r-1) + b \cdot (n-r)^2.$$

Notice that, the r could be anywhere between 1 and n; hence,

$$T_W(n) \le \max_{1 \le r \le n} a \cdot n + T_W(r-1) + b \cdot (n-r)^2.$$

We guess a solution with $T_W(n) = O(n^2)$. That is, $T_W(n) \le c \cdot n^2$ for some c. Then, to check the solution, we have

$$T_W(n) \le \max_{1 \le r \le n} a \cdot n + c \cdot (r-1)^2 + b \cdot (n-r)^2.$$

We use $f(r)$ to denote the RHS. Notice that f is convex and reaches its maximum at one of the two end points (you need check this by taking derivative (abuse math here) over r in the RHS). Therefore, the maximum of the RHS is the maximum of the RHS with $r = 1$ and $r = n$ (two end points). That is,

$$T_W(n) \le a \cdot n + \max(b \cdot (n-1)^2, c \cdot (n-1)^2).$$

Hence, by taking c large (and also n large),

$$T_W(n) \le c \cdot n^2.$$

Therefore, the worst case complexity of mix sort is $O(n^2)$. In fact, the worst case complexity is $\Theta(n^2)$ since the compelxity is reached when the input array starts with the minimal element followed by a strictly decreasing subarray.

Finally, we will consider the average case. First, we fix an r – the position that divides the low/high parts. Notice that the low-part and the high-part are still random after the partition. Hence,

$$T_{avg}(n) = \Theta(n) + T_{avg}(r-1) + T_{avg}^{\text{insertsort}}(n-r).$$

We let the $\Theta(n) \leq a \cdot n$ for some constant a. Since the average compelxity of insertsort $T_{avg}^{\text{insertsort}}(n-r) = O((n-r)^2) \leq b \cdot (n-r)^2$, for some constant b, we have

$$T_{avg}(n) \leq a \cdot n + T_{avg}(r-1) + b \cdot (n-r)^2.$$

Now, we make r random, i.e.,

$$T_{avg}(n) \leq \sum_{1 \leq r \leq n} \frac{1}{n} \cdot (a \cdot n + T_{avg}(r-1) + b \cdot (n-r)^2).$$

That is,

$$T_{avg}(n) \leq a \cdot n + b \cdot \frac{1}{n} \cdot \sum_{1 \leq r \leq n} (n-r)^2 + \frac{1}{n} \cdot \sum_{1 \leq r \leq n} T_{avg}(r-1).$$

First, we have $b \cdot \frac{1}{n} \cdot \sum_{1 \leq r \leq n} (n-r)^2 \leq \frac{b(n-1)^3}{3n}$ by taking integral and abusing math. Then, we have

$$T_{avg}(n) \leq a \cdot n + \frac{b(n-1)^3}{3n} + \frac{1}{n} \cdot \sum_{1 \leq r \leq n} T_{avg}(r-1).$$

Now, we guess a solution $T_{avg}(n) = O(n^2)$; i.e., $T_{avg}(n) \leq c \cdot n^2$ for some c. To check the solution, we have,

$$T_{avg}(n) \leq a \cdot n + \frac{b(n-1)^3}{3n} + \frac{1}{n} \cdot \sum_{1 \leq r \leq n} c \cdot (r-1)^2.$$

Notice that $\sum_{1 \leq r \leq n} (r-1)^2 \leq \frac{(n-1)^3}{3}$. Therefore,

$$T_{avg}(n) \leq a \cdot n + \frac{b(n-1)^3}{3n} + \frac{1}{n} \cdot c \cdot \frac{(n-1)^3}{3}.$$

Furth simplying the RHS, we have

$$T_{avg}(n) \leq a \cdot n + \frac{bn^2}{3} + \frac{cn^2}{3}.$$

Taking c large (and also n large), we have

$$T_{avg}(n) \leq cn^2.$$

Hence, the average case complexity of mixsort is $O(n^2)$.