CS3230
Tutorial 3

1. Draw a decision tree to show that 4 numbers can be sorted using at most 5 comparisons.

   Ans:

   (i). Suppose the four numbers are a, b, c, d.

   (ii). First compare a, b. Then compare, c and d.

   As all possible answers are symmertical, assume $a \leq b$ and $c \leq d$ for the following.

   (iii). Compare a and c.

   (iv). In (iii), if $a \leq c$: compare b with c and d.

   > If $b \leq c$, then the order is $a \leq b \leq c \leq d$.
   > If $b > c$ but $b \leq d$, then the order is $a \leq c \leq b \leq d$.
   > Otherwise, the order is $a \leq c \leq d \leq b$.

   (v). In (iii), if $c < a$, then compare d with a and b.

   > If $d \leq a$, then the order is $c \leq d \leq a \leq b$.
   > If $d > a$ but $d \leq b$, then the order is $c \leq a \leq d \leq b$.
   > Otherwise, the order is $c \leq d \leq b \leq a$.

2. Consider the following modification of the partition algorithm done in class. Show that it works correctly.

   Partition(A, i, j)
   Assumption: $i < j$
   1. Let $m = i + 1$, $n = j$;
   2. While $m \leq n$, do {
      3. While $A[m] < A[i]$ and $m \leq n$ do { $m = m + 1$ }
      4. While $A[n] \geq A[i]$ and $n \geq m$ do { $n = n - 1$ }
      5. If $n > m$, then swap($A[m]$,$A[n]$).
      }
   6. swap$(A[i], A[n])$
   7. Return n.

   End

Ans:

First note that m is monotonically non-decreasing and n is monotonically non-increasing throughout the algorithm.

Invariants of the algorithm during the while loop (steps 2–5)

I1: $A[i+1], \ldots, A[m-1]$, are $< A[i]$
I2: $A[n+1], \ldots, A[j]$, are $\geq A[i]$

Clearly, the invariants hold at the beginning of the first iteration of the while loop at step 2, as at that time $m = i+1$ and $n = j$.

Furthermore, whenever value of m or n is changed in steps 3 and 4, the invariants are maintained.

Also,

While loop at step 3, increases m until $A[m] \geq A[i]$ or m becomes $> n$.
Thus, at the end of the while loop at step 3, either $m > n$, or $A[m] \geq A[i]$.
Similarly, while loop at step 4, decreases n until $A[n] < A[i]$, or $n < m$.
Thus, at the end of the while loop at step 3, either $m > n$, or $A[n] < A[i]$.

Thus, at the end of any iteration of the while loop (of step 2), either $m > n$, or ($A[m] < A[i]$ and $A[n] \geq A[i]$) — in the later case, in the next iteration of the while loop of step 2, m will increase and n will decrease.

Thus, m increases in value after every iteration of the while loop in step 2 (except maybe for the first iteration), and n decreases in value after very iteration of the while loop in step 2 (except maybe for the first iteration).

It follows that eventually, $m > n$ and the while loop at step 2 will terminate. Using the invariants, we immediately have that at this point, $m = n+1$, $A[i+1], \ldots, A[n]$ are $< A[i]$ and $A[m], \ldots, A[j]$ are $\geq A[i]$.

Thus, step 6 correctly places the pivot value at $A[n]$, and at the end, $A[i], \ldots, A[n-1]$ are smaller than $A[n]$, and $A[n+1], \ldots, A[j]$ are $\geq A[n]$, as needed.

3. Suppose an array A contains n numbers between 1 and m. Write an algorithm with worst case complexity of $O(m+n)$, which finds the median element of the array.

   Ans:

   1. Compute $C[i]$ as the number of elements in the array which are $\leq i$. This can be done in $O(m+n)$ time by first counting the number of elements, $M[i]$ which are equal to i (in time $O(n)$), and then taking the cumulative sum of the array M (in time $O(m)$).

   2. Find least i such that $C[i] \geq \frac{n}{2}$. Return i. This step takes $O(m)$ time.

4. In merge sort algorithm done in class, for merging two arrays of size m and n respectively, we used one extra array of size $m+n$.

   Can you give an algorithm to do the merge which uses an extra array B of size only $\min(m, n)$? Complexity of your algorithm should still be linear in the size of the two arrays.

   Ans: Below is the main algorithm.

Merge(A, i, k, j)

Assumption: $i \leq k < j$; $A[i:k]$ and $A[k+1:j]$ are sorted; and $k - i \leq j - k - 1$ (that is $A[i:k]$ is smaller in size than $A[k+1:j]$)

1. For $r = i$ to k do $B[r] = A[r]$ Endfor
   (* First copy the smaller section to be merged into B *)
2. $p1 = i$; $p2 = k + 1$; $p3 = i$
3. While $p1 \leq k$, $p2 \leq j$ do
   3.1 If $B[p1] \leq A[p2]$, then $A[p3] = B[p1]$, $p1 = p1 + 1$
   Else $A[p3] = A[p2]$, $p2 = p2 + 1$; Endif
   3.2 $p3 = p3 + 1$;
   Endwhile
4. While $p1 \leq k$, do
   $A[p3] = B[p1]$; $p1 = p1 + 1$; $p3 = p3 + 1$;
   Endwhile
   (* Note: we don't need to do the following part
   While $p2 \leq j$ do
   $A[p3] = A[p2]$; $p2 = p2 + 1$; $p3 = p3 + 1$;
   Endwhile
   as at this point $p2 = p3$. *)

End

Basically, the above construction is similar to what we did in class, except that, after copying in step 1, we are merging $B[i..k]$ with $A[k+1..j]$. The role of $B[i..j]$ in the algorithm done in class is now played by $A[i..j]$. So the only thing we need to ensure is that at any point of time $p3$ does not become larger than $p2$. To see this, note that initially the difference $p2 - p3$ is $k + 1 - i$. Furthermore, $p3$ is increased without increasing $p2$ at most $k + 1 - i$ times (due to step 3 or step 4). Thus, $p2 \geq p3$ holds throughout the algorithm.

5. A tromino is formed by taking away one 1×1 square from a 2×2 square.

   (a) Show that any board of size $2j \times 3k$ (without any missing squares) can be tiled using trominoes.

   (b) Show that a 5×5 board, with a corner missing, can be tiled using trominos.

   (c) Show that any 7×7 board with any one square missing can be tiled using trominos.

   (This is lengthy, and done by considering several possible cases of missing square. So if you can't find an answer just assume that this can be done).

   (d) Show that a 11×11 board, with any one square missing can be tiled using trominos.

   Hint: Divide the board into a sub-board of size 7×7 (which contains that missing square), a sub-board of size 5×5 with a corner missing, and two sub-boards of size 4×6.

   (e) By induction show that, for $n > 11$, any $n \times n$ board with one square missing can be tiled using trominoes as long as n is odd and not a multiple of 3.

   Hint: Divide the board into four parts, One of size $(n - m) \times (n - m)$ containing the missing square (which can be solved using induction), two boards of sizes $(n - m - 1) \times m$,

which can be solved using part (a), and one board of size $(m+1)\times(m+1)$, with a missing corner.

Ans:

(a) Note that two trominoes put together can cover a 2×3 square. By using this repeatedly, we can cover any $2j \times 3k$ board.

(b) See Tiling 1.

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

Tiling 1

(c) Consider the two partial tilings given in Tiling 2 and Tiling 3, where the missing square may be one of ?. Note that all possible places for missing squares are symmetrical to one of these cases. Furthermore, in each of these cases it is easy to tile the remaining left-over squares once the position missing square is known.

1	1	5	5	7	7	8
1	2	5	6	7	8	8
2	2	6	6	10	10	11
3	3	4	9	10	11	11
3	4	4	9	9	12	l2
?	?		?	?	12	13
?	?		?	?	13	13

Tiling 2

1	1	2	10	10	7	7
1	2	2	9	10	7	8
3	3	9	9	11	8	8
3	4	?	?	11	11	12
4	4	?	?	14	12	12
5	6	6	15	14	14	13
5	5	6	15	15	13	13

Tiling 3

(d) Suppose the original board was $A[1:11, 1:11]$.

Suppose without loss of generality that the missing square belongs to $A[1:7, 1:7]$ (any other missing square can be handled in a symmetric way).

Then tile $A[1:7, 1:7]$ using part (c). Tile $A[7:11, 7:11]$, with missing square $A[7, 7]$ as in part (b). Tile, $A[1:6, 8:11]$ and $A[8:11, 1:6]$ using part (a).

(e) Suppose the original board was $A[1:n, 1:n]$.

Suppose without loss of generality that the missing square belongs to $A[1:n-6, 1:n-6]$.

Consider the sub-boards $A[1:n-6, 1:n-6]$ (which is tiled inductively), $A[1:n-7, n-5:n]$, $A[n-5:n, 1:n-7]$ (which can be tiled using part (a)), $A[n-6:n, n-6:n]$ (with missing corner, which can be tiled using part (c)).

6. Suppose we are sorting an array consisting of integers ≥ 0. Explain how one may modify the inputs (in linear time) so that any sorting algorithm becomes a stable sorting algorithm.

   Your modification of the numbers should be such that one can easily obtain the original numbers back from the modification.

   Ans: (A) Suppose there are n integers in the array A.

   For $i = 1$ to n: let $A[i] = A[i] * (n+1) + i$.

   (B) Sort the array A

   (C) For $j = 1$ to n: let $A[j] = \lfloor \frac{A[j]}{n+1} \rfloor$.