Tutorial 7: Searching and Graphs
Example Solutions

(1) (a) i. By the definition of the median and the way that partition works it is clear that the lower half has size $\lfloor n/2 \rfloor$ and the upper half has size $\lceil n/2 \rceil$.

ii. The recurrence here is simple:

$$T(n) = \begin{cases} \Theta(1), & \text{if } n \leq 1; \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + M(n) + \Theta(n), & \text{if } n \geq 2. \end{cases}$$

iii. Here $M(n) + \Theta(1) = \Theta(n)$. So in the Master Theorem the critical exponent $e = \log_2(2) = 1$. Thus the middle case applies and we have $T(n) = \Theta(n \lg n)$.

iv. See the chapter on on *Medians and Order Statistics* in [CLRS]. The algorithm is not particularly complicated (around 2 pages for a careful description and analysis including a diagram).

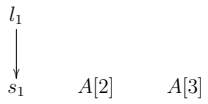
(2) The following algorithm sortFour does the job (mergeSort also sorts 4 numbers with 5 comparisons).

```
sortFour(A)
  (s1, ℓ1) ← comp(A[0], A[1]).
  (s2, ℓ2) ← comp(A[2], A[3]).
  (s3, ℓ3) ← comp(s1, s2).
  (s4, ℓ4) ← comp(ℓ1, ℓ2).
  (s5, ℓ5) ← comp(ℓ3, s4)
  return ℓ4, ℓ5, s5, s3
```

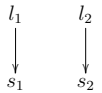
We can discover this algorithm as follows. We display elements and their relationship (in terms of relative order) at each stage. An arrow from a to b means that we now know $a > b$. Initially we know nothing about the ordering of the given elements.

$A[0]$ $A[1]$ $A[2]$ $A[3]$

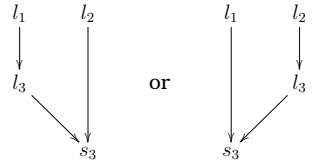
Compare $A[0]$ with $A[1]$



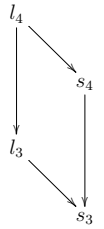
Compare $A[2]$ with $A[3]$



Compare s_1 with s_2



Compare l_1 with l_2



Compare l_3 with s_4

We have now finished.

This is best possible: 5 comparisons is the smallest number for the general case of sorting 4 elements. To see why this is the case, we use the fact that for general inputs of 4 numbers, there are 24 possible different rearrangements of those numbers which correspond to possible outputs (because there are $4!$ permutations of any 4 items). We can view sorting algorithms as binary decision trees, where vertices are comparisons. As the algorithm is executed on an input we follow a path from the root of the tree to a leaf (so the maximum number of comparisons made is the height of the tree). Each comparison leads us to one of two possibilities. By the time we get to a leaf of the tree (all comparisons have been made for this input) the ordering is fixed. Thus there must be at least as many leaves as there are possible orderings of the input. Since a binary tree of height h has at most 2^h leaves we must have $h \geq \lg(24) > 4$.

This reasoning can be generalised to the case of n items. We see that we must use at least $\lg(n!)$ comparisons. Since $(n/2)^{n/2} < n! \leq n^n$ we deduce that $\Omega(n \lg n)$ comparisons are necessary.

(3) (a) A single edge is a bipartite graph and a triangle is not.

(b) We use a search of the graph (e.g., BFS) with the adjacency list representation. We maintain a variable V that alternates between 1 and 2 (to denote V_1 or V_2), intialise the variable to 1 (or 2, it doesn't matter). Each time we visit a vertex we first give V the alternative value (i.e., alternate between 1 and 2). When we visit a vertex for the first time we mark it with the vertex set denoted by V. When we revisit a vertex we check if the set indicated by V is consistent with the one recorded. If it is carry on otherwise report that the graph is not bipartite.

Since we do only a constant amount of work at each vertex the time is asymptotically the same as for a straight search, i.e., $\Theta(n+m)$.