Sorting (chapter 7)
1. P144 5-2 (g-i), for (h) only consider k=0 and k=1.
2. p173 7.1-1
3. p178 7.2-2
4. Programming assignment 1 (50% weight of ADw5)

P144 5-2 (g)

Since DETERMINISTIC-SEARCH would search through the entire array in the event of not finding x, both average-case and worst-case would be $\Theta(n)$.

P144 5-2 (h)

Assume search cost dominates permutation cost, i.e. $g(n) = o(f(n))$ where $f(n)$ denotes search cost and $g(n)$ permutation cost.

k = 0
Both average-case and worst-case would be $\Theta(n)$.

k = 1
Assume searching each index incurs constant cost C.
Average-case: $T(n) = C(n+1)/2 + g(n) = \Theta(n)$
Worst-case: $T(n) = Cn + g(n) = \Theta(n)$

P144 5-2 (i)

DETERMINISTIC-SEARCH is obviously better than RANDOM-SEARCH because 1. it would not check the same index more than once, and 2. it is guaranteed to terminate.

If the underlying distribution of A is heavily skewed towards the worst-case for DETERMINISTIC-SEARCH, i.e. x towards the end of A, SCRAMBLE-SEARCH might be better, if the cost of scrambling is less than the cost of having to search through the whole array A. In practice, however, in the time it takes to scramble A, we might as well have done a DETERMINISTIC-SEARCH.

So DETERMINISTIC-SEARCH is likely to be the best.

p173 7.1-1

Following Figure 7.1, we use the last element 11 as pivot. We let p denote the index of the first element, and r the index of the last element.

i	p j											r
	13	19	9	5	12	8	7	4	21	2	6	11
i	p	j										r
	13	19	9	5	12	8	7	4	21	2	6	11
i	p		j									r
	13	19	9	5	12	8	7	4	21	2	6	11
	p i			j								r
	9	19	13	5	12	8	7	4	21	2	6	11
	p	i			j							r
	9	5	13	19	12	8	7	4	21	2	6	11
	p	i				j						r
	9	5	13	19	12	8	7	4	21	2	6	11

p		i				j					r
9	5	8	19	12	13	7	4	21	2	6	11

p			i				j				r
9	5	8	7	12	13	19	4	21	2	6	11

p				i				j			r
9	5	8	7	4	13	19	12	21	2	6	11

p				i					j		r
9	5	8	7	4	13	19	12	21	2	6	11

p					i					j	r
9	5	8	7	4	2	19	12	21	13	6	11

p						i					j r
9	5	8	7	4	2	6	12	21	13	19	11

p						i					j r
9	5	8	7	4	2	6	11	21	13	19	12

p178 7.2-2

The function PARTITION(*A,p,r*) on page 171 is reproduced below for complexity analysis.

```
1.  PARTITION (A, p, r):
2.      x = A[r]
3.      i = p-1
4.      for j=p to r-1:
5.          if A[j] ≤ x:
6.              i = i+1
7.              exchange A[i] with A[j]
8.      exchange A[i+1] with A[r]
9.      return i+1
```

Let the constant cost in line 2, 3, 8 & 9 be denoted by C_1, and the cost in line 6 & 7 be C_2. Since the if condition in line 5 is always fulfilled, lines 6 & 7 would be executed for (r-1)-(p-1) = r-p = n-1 times. Therefore the cost of PARTITION(*A,p,r*) is

$$T_{partition}(n) = (n-1)C_2 + C_1$$

Every run of PARTITION(*A,p,r*) returns i+1=r, meaning each time QUICKSORT needs to perform PARTITION(*A,p,r*) on an array with one fewer element. Therefore,

$$T_{quicksort}(n) = T_{quicksort}(n-1) + T_{partition}(n)$$

$$T_{quicksort}(N) = \sum_{n=1}^{N} (n-1)C_2 + C_1$$

$$T_{quicksort}(N) = (N-1)\frac{N}{2}C_2 + NC_1 = \Theta(N^2)$$