Homework 1 (70 points), Spring 2004

Q1: Exercise 2.1-2 (10 points)

Rewrite the Insertion-Sort procedure to sort into nonincreasing instead of nondecreasing order.

Insertion-Sort(A)
1. for j <-- 2 to length[A]
2. do key <-- A[j]
3. // Insert A[j] into the sorted sequence A[1..j-1]
4. i <-- j-1
5. while i>0 and A[i]<key
7. i <-- i-1
8. A[i+1] <-- key

Q2: Exercise 2.2-1 (10 points)

Express the function $n^3/1000 - 100n^2 - 100n + 3$ in terms of $\Theta$-notation.

- An easy way to determine the $\Theta$-notation for a function – pick the most dominant term without considering its coefficient. Thus, $n^3/1000 - 100n^2 - 100n + 3 = \Theta(n^3)$

Q3: Exercise 2.3-2 (10 points)

Rewrite the Merge procedure so that it does not use sentinels, instead stopping once either array $L$ or $R$ has had all its elements copied back to $A$ and then copying the remainder of the other array back into $A$.

- Pseudocode for Merger sort without sentinels.

Merge(A, p, q, r)
1. n1 <-- q - p + 1
2. n2 <-- r - q
3. create arrays L[1..n1] and R[1..n2]
4. for i <-- 1 to n1
5. do L[i] <-- A[p+i-1]
6. for j <-- 1 to n2
7. do R[j] <-- A[q+j]
8. i <-- 1
9. j <-- 1
10. for k <-- p to r
11. do if (i > n1)
12. then A[k] <-- R[j]
13. j <-- j+1
14. else if (j > n2)
15. then A[k] <-- L[i]
16. i <-- i+1
17. else if (L[i] <= R[j])
18. then A[k] <-- L[i]
19. i <-- i+1
20. else A[k] <-- R[j]
21. j <-- j+1
Q4: Exercise 2.3-4 (10 points)

Insertion sort can be expressed as a recursive procedure as follows. In order to sort \( A[1..n] \), we recursively sort \( A[1..n-1] \) and then insert \( A[n] \) into the sorted array \( A[1..n-1] \). Write a recurrence for the running time of this recursive version of insertion sort.

- **Divide:** Divide the problem into one smaller subproblem with size \( n-1 \).
- **Conquer:** Conquer the subproblem by solving it recursively. After this step, we have a sorted subarray with size \( n-1 \).
- **Combine:** Insert the element not in the sorted subarray into the sorted subarray.

Therefore, the recurrence for this version of Insertion-Sort is as follows.

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1 \\
T(n-1) + O(n) & \text{if } n > 1 
\end{cases}
\]

Q5: Problem 2-1 parts a, b and c (10 points)

**Insertion sort on small arrays in merge sort**

Although merge sort runs in \( \Theta(n \lg n) \) worst-case time and insertion sort runs in \( \Theta(n^2) \) worst-case time, the constant factors in insertion sort make it faster for small \( n \). Thus, it makes sense to use insertion sort within merge sort when subproblems become sufficiently small. Consider a modification for merge sort in which \( n/k \) sublists of length \( k \) are sorted using insertion sort and then merged using the standard merging mechanism, where \( k \) is a value to be determined.

a. Show that the \( n/k \) sublists, each of length \( k \), can be sorted by insertion sort in \( \Theta(nk) \) worst-case time.
   
   - Each sublist with length \( k \) takes \( \Theta(k^2) \) worst-case time using Insertion-Sort. To sort \( n/k \) such sublists, it takes \( n/k \times \Theta(k^2) = \Theta(nk) \) worst-case time.

b. Show that the sublists can be merged in \( \Theta(n \lg(n/k)) \) worst-case time.
   
   - Merging \( n/k \) sublists into \( n/2k \) sublists takes \( \Theta(n) \) worst-case time.
   - Merging \( n/2k \) sublists into \( n/4k \) sublists takes \( \Theta(n) \) worst-case time
   - .......
   - Merging 2 sublists into one list takes \( \Theta(n) \) worst-case time
   - We have \( \lg(n/k) \) such merges, so merging \( n/k \) sublists into one list takes \( \Theta(n \lg(n/k)) \) worst-case time.

c. Given that the modified algorithm runs in \( \Theta(nk + n \lg(n/k)) \) worst-case time, what is the largest asymptotic (\( \Theta \)-notation) value of \( k \) as a function of \( n \) for which the modified algorithm has the same asymptotic running time as standard merge sort?
   
   - In order for \( \Theta(nk + n \lg(n/k)) = \Theta(n \lg n) \), the largest asymptotic value for \( k \) is \( \Theta(\lg n) \).

Q6: (10 points)

Please use the basic definition of \( \Theta \)-notation to prove \( n^2/3 - 12 = \Theta(n^2) \).
• We would like to find positive constants \( c_1, c_2 \) and \( n_0 \), such that

\[
0 \leq c_1 n^2 \leq n^2/3 - 12 \leq c_2 n^2, \forall n \geq n_0
\]

Such constants do exist, for example, \( c_1 = 1/9, c_2 = 1/3 \) and \( n_0 = 9 \)
Therefore, \( n^2/3 - 12 = \Theta(n^2) \)

**Q7: Exercise 3.1-4 (10 points)**

Is \( 2^{n+1} = O(2^n) \)? Is \( 2^{2n} = O(2^n) \)?

• We can choose \( c = 2 \) and \( n_0 = 0 \), such that \( 0 \leq 2^{n+1} \leq c \times 2^n \) for all \( n \geq n_0 \). By definition, \( 2^{n+1} = O(2^n) \).

• We can not find any \( c \) and \( n_0 \), such that \( 0 \leq 2^{2n} = 4^n \leq c \times 2^n \) for all \( n \geq n_0 \). Therefore, \( 2^{2n} \neq O(2^n) \).