

Assignment 3 Cover Page

Due: Monday, March 22 at 12:00 noon

- You may choose to work in a group of two, and submit a single assignment report.
- Please read “How to Prepare Assignment Reports” from the course web page.
- Please make your answers clear and succinct.
- Please use this page as the cover page for your assignment.

Student 1:

Family Name:

Given Name:

Student #:

Email:

Student 2:

Family Name:

Given Name:

Student #:

Email:

Section to which the assignment will be returned (circle one):

- M (TR11:30-13:00)
- N (W19:00-22:00)
- P (WF16:00-17:30)

Problem	Marking Scheme	Score
1	10 marks	
2	20 = 10 + 5 + 5 marks	
3	35 = 5 + 15 + 15 marks	
4	35 = 10 + 15 + 10 marks	
Total	100 marks	

Assignment 3

Due: Monday, March 15 at 12:00 noon

Please read “How to Prepare Assignment Reports” from the course web page.

1. **Sorting variable-length items**, CLRS 8-3b. [10]

This problem deals with linear-time sorts, such as counting sort or radix sort. You are given an array of strings, where different strings may have different numbers of characters, but the total number of characters overall the strings is n . Show how to sort the strings in the lexicographical order (that is, $a < ab < b$) in $O(n)$ time.

2. **Building a heap using insertion**, adapted from CLRS 6-1 [20]

The algorithm BUILD-MAX-HEAP described in CLRS builds a heap with the aid of the routine MAX-HEAPIFY. An alternate design (BUILD-MAX-HEAP'), uses MAX-HEAP-INSERT (see code on next page).

(a) Do the procedures BUILD-MAX-HEAP and BUILD-MAX-HEAP' always [10]
create the same heap when run on the same input array? Prove that they do, or provide a counterexample.

(b) What is the best-case running time for the two methods? Justify your answer. [5]

(c) What is the worst-case running time for the two methods? Justify your answer. [5]

```

BUILD-MAX-HEAP(A)
  heap-size[A] = length[A]
  for i = ⌊length[A]/2⌋ downto 1
    MAX-HEAPIFY(A,i)

```

```

BUILD-MAX-HEAP'(A)
  heap-size[A] = 1
  for i = 2 to length[A]
    MAX-HEAP-INSERT(A, A[i])

```

```

MAX-HEAPIFY(A, i)
  l = LEFT(i)
  r = RIGHT(i)
  if l ≤ heap-size[A] and A[l] > A[i]
    largest = l
  else
    largest = i
  if r ≤ heap-size[A] and A[r] > A[largest]
    largest = r
  if largest ≠ i
    exchange A[i] ↔ A[largest]
    MAX-HEAPIFY(A, largest)

```

```

MAX-HEAP-INSERT(A, key)
  heap-size[A] = heap-size[A] + 1
  A[heap-size[A]] = −∞
  HEAP-INCREASE-KEY(A, heap-size[A], key)

```

```

HEAP-INCREASE-KEY(A, i, key)
  if key < A[i]
    error "new key is smaller than current key"
  A[i] = key
  while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] ↔ A[PARENT(i)]
    i = PARENT(i)

```

3. 2-location activity scheduling

[35]

Consider the problem of scheduling a set of “activities”, each with a start time and finish time, but now instead of scheduling them in one location, you need to schedule them in two locations. Think, for example, of scheduling lectures in two rooms: no two lectures in the same room can overlap, but there can be simultaneously lectures in both rooms; also, each lecture only needs to be scheduled in one of the rooms, not both.

The input is a list of activities $(s_1, f_1) \dots (s_n, f_n)$, where s_i is a starting time of activity i and f_i is a finishing time of activity i . A feasible schedule is a pair (A_1, A_2) such that two conditions hold:

- (a) If for activities i and j both $i \in A_1$ and $j \in A_1$ or both $i \in A_2$ and $j \in A_2$, then activity i does not overlap activity j (that is, $f_i \leq s_j$ or $f_j \leq s_i$).
- (b) $A_1 \cap A_2 = \emptyset$, that is, no activity is scheduled in both A_1 and A_2 .

The goal is to compute a feasible schedule that includes as many activities as possible.

- (a) Give an example of a set of activities that gives rise to at least two possible optimal schedules, and that has an activity that does not get scheduled in either case. [5]
- (b) Give a polynomial-time greedy algorithm to solve this problem. [15]
- (c) Prove that your algorithm always returns an optimal schedule. [15]

4. Longest Palindrome

[35]

The input is a string $\bar{s} = s_1 \dots s_n$. Your goal is to find a maximum-length palindrome in s , that is, a subsequence of s that reads the same forward and backward.

For example, if your input string is “initialization”, then a possible longest palindrome is “n i t a z a t i n”, another “n i t i a i t i n”, both with length 9.

- (a) Define a suitable array A to store partial solutions to the problem and state how to find from this table the *length* of the longest palindrome. [10]
- (b) Give a recurrence to compute elements of A , including initialization. [15]
- (c) Explain how to find one of the actual longest palindromes given A . [10]