Assignment 4

**R-2.8 Illustrate the performance of the selection-sort algorithm on the following input sequence (22, 15, 26, 44, 10, 3, 9, 13, 29, 25).**

Answer:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 22 | 15 | 26 | 44 | 10 | 3 | 9 | 13 | 29 | 25 |
| 3 | 15 | 26 | 44 | 10 | 22 | 9 | 13 | 29 | 25 |
| 3 | 9 | 26 | 44 | 10 | 22 | 15 | 13 | 29 | 25 |
| 3 | 9 | 10 | 44 | 26 | 22 | 15 | 13 | 29 | 25 |
| 3 | 9 | 10 | 13 | 26 | 22 | 15 | 44 | 29 | 25 |
| 3 | 9 | 10 | 13 | 15 | 22 | 26 | 44 | 29 | 25 |
| 3 | 9 | 10 | 13 | 15 | 22 | 26 | 44 | 29 | 25 |
| 3 | 9 | 10 | 13 | 15 | 22 | 25 | 44 | 29 | 26 |
| 3 | 9 | 10 | 13 | 15 | 22 | 25 | 26 | 29 | 44 |
| 3 | 9 | 10 | 13 | 15 | 22 | 25 | 26 | 29 | 44 |

**R-2.9 Illustrate the performance of the insertion-sort algorithm on the input sequence of the previous problem.**

Answer:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 22 | 15 | 26 | 44 | 10 | 3 | 9 | 13 | 29 | 25 |
| 15 | 22 | 26 | 44 | 10 | 3 | 9 | 13 | 29 | 25 |
| 15 | 22 | 26 | 44 | 10 | 3 | 9 | 13 | 29 | 25 |
| 15 | 22 | 26 | 44 | 10 | 3 | 9 | 13 | 29 | 25 |
| 10 | 15 | 22 | 26 | 44 | 3 | 9 | 13 | 29 | 25 |
| 3 | 10 | 15 | 22 | 26 | 44 | 9 | 13 | 29 | 25 |
| 3 | 9 | 10 | 15 | 22 | 26 | 44 | 13 | 29 | 25 |
| 3 | 9 | 10 | 13 | 15 | 22 | 26 | 44 | 29 | 25 |
| 3 | 9 | 10 | 13 | 15 | 22 | 26 | 29 | 44 | 25 |
| 3 | 9 | 10 | 13 | 15 | 22 | 25 | 26 | 29 | 44 |

**R-2.10 Give an example of a worst-case sequence with n elements for insertion-sort runs in Ω(n2) time on such a sequence.**

Answer:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 90 | 80 | 70 | 60 | 50 | 40 | 30 | 20 | 10 |

**R-2.13 Suppose a binary tree T is implemented using a vector S, as described in Section 2.3.4. If n items are stored in S in sorted order, starting with index 1, is the tree T a heap? Justify your answer.**

Answer: Yes, it is

For example:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 3 | 9 | 10 | 13 | 15 | 22 | 25 | 26 | 29 | 44 |

A picture containing diagram, line, circle, white

Description automatically generated

**R-2-18 Draw an example of a heap whose keys are all the odd numbers from 1 to 49 (with no repeats), such that the insertion of an item with key 32 would cause up-heap bubbling to proceed all the way up to a child of the root (replacing that child’s key with 32).**

Answer:

A picture containing white, circle

Description automatically generated

**C-2.32 Let T be a heap storing n keys. Give an efficient algorithm for reporting all the keys in T that are smaller than or equal to a given query key x (which is not necessarily in T). For example, given the heap on Figure 2.41 and query key x=7, the algorithm should report 4, 5, 6, 7. Note that the keys do not need to be reported in sorted order. Ideally, your algorithm should run in O(k) time, where k is the number of keys reported.**

Answer:

Algorithm find\_smaller(T, x)

S 🡨 EmptySequence

find\_helper(T, x, T.root(), S)

return S

Algorithm find\_helper(T, x, p, S)

e 🡨 p.element()

if e > x then

return

S.insertLast(e)

if T.isExtra(p) then

return

find\_helper(T, x, T.leftChild(p), S)

find\_helper(T, x, T.rightChild(p), S)

**Design an algorithm, isPermutation(A,B) that takes two sequences A and B and determines whether or not they are permutations of each other, i.e., same elements but possibly occurring in a different order. Hint: A and B may contain duplicates.**

**What is the worst case time complexity of your algorithm? Justify your answer.**

Answer:

Algorithm is\_permulation(A, B) {

if A.size() != B.size() then 1

return false 1

n 🡨 A.size() 1

for r = 1 to n-1 do { n

is\_valid 🡨 false n

p 🡨 A.atRank(r) n

for i = r to n-1 do { n2

q 🡨 B.rankAt(i) n2

if q.element() == p.element() then { n2

B.swapElements(B.atRank(r),q) n2

is\_valid = true n2

break n2

}

}

if !is\_valid then n

return false n

}

return true 1

}

The worst case time complexity is: T(n2)