Jared Yu

EN.685.621

Homework 1

Self-Study Problems

2.2-2

```
SELECTION-SORT(A)                   cost     times
1. for j = 1 to (A.length – 1)      $c_1$    $n-1$
2.   min = A[j]                     $c_2$    $n-1$
3.   for i = (j + 1) to A.length    $c_3$    $\big(n(n-1)\big)/2$
4.     if A[i] < min                $c_4$    $\sum_{j=1}^{n-1} t_j$
5.       temp = min                 $c_5$    $\sum_{j=1}^{n-1} t_j$
6.       min = A[i]                 $c_6$    $\sum_{j=1}^{n-1} t_j$
7.       A[i] = temp                $c_7$    $\sum_{j=1}^{n-1} t_j$
8.   A[j] = min                     $c_8$    $n-1$
```

Where $n = A.length$, t_j is the number of times the if statement in line 4 executes for that value of j.

Loop invariant: At the start of each iteration of the for loop of lines 1-8, the subarray A[1…j] consists of the ordered minimum values of the array.

By the (n-1) loop, the nth element will be sorted at the same time.

Best case: Already sorted array, lines 4-7 are skipped.

$$\begin{aligned} T(n) &= c_1(n-1) + c_2(n-1) + c_3((n(n-1))/2) + c_8(n-1) \\ &= (n-1)\left(c_1 + c_2 + \frac{(c_3 n)}{2} + c_8\right); \Theta(n^2) \end{aligned}$$

Worst case: Array sorted backwards so that lines 4-7 will run the maximum number of times.

$$
\begin{aligned}
T(n) = c_1(n-1) + c_2(n-1) + c_3\left(\frac{\left(n(n-1)\right)}{2}\right) + c_4\left(\frac{\left(n(n-1)\right)}{2}\right) + c_5\left(\frac{\left(n(n-1)\right)}{2}\right) \\
+ c_6\left(\frac{\left(n(n-1)\right)}{2}\right) + c_7\left(\frac{\left(n(n-1)\right)}{2}\right) + c_8(n-1) \\
= (n-1)\left[c_1 + c_2 + \frac{(c_3 n)}{2} + \frac{(c_4 n)}{2} + \frac{(c_5 n)}{2} + \frac{(c_6 n)}{2} + \frac{(c_7 n)}{2} + c_8\right] \\
= (n-1)\left[c_1 + c_2 + c_8 + \left(\frac{n}{2}\right)(c_3 + c_4 + \cdots + c_7)\right]
\end{aligned}
$$

$$= (n-1)\left[a + \left(\frac{n}{2}\right)b\right]$$

$$where\ a = c_1 + c_2 + c_8, and\ b = c_3 + c_4 + \cdots + c_7$$

$$= (n-1)a + (n-1)\left(\frac{n}{2}\right)b$$

$$= (n-1)a + \frac{(n^2 - n)b}{2}$$

$$= na - a - \frac{n^2 b}{2} - \frac{nb}{2}$$

$$= \Theta(n^2)$$

2.3-5

https://www.quora.com/What-is-the-time-complexity-of-binary-search-algorithm/answer/Daniel-R-Page

https://www2.cs.arizona.edu/~mercer/Presentations/12-AlgorithmAnalysis.pdf

(also used previous notes from Data Structures professor in undergraduate university)

BINARY-SEARCH(A, v)	cost	times
1. lo = 1	c_1	1
2. hi = A.length	c_2	1
3. while lo <= hi:	c_3	$\frac{n}{2}$
4. mid = lo + floor((hi - lo) / 2)	c_4	$\frac{n}{2}$
5. if A[mid] == v:	c_5	0 or 1
6. return mid	c_6	0 or 1
7. if A[mid] < v:	c_7	$\frac{n}{2}$

8. lo = mid + 1	c_8	$\frac{n}{2}$
9. else	c_9	$\frac{n}{2}$
10. hi = mid – 1	c_{10}	$\frac{n}{2}$
11. return "Not found"	c_{11}	0 or 1

If found:

$$T(n) = c_1(1) + c_2(1) + c_3\left(\frac{n}{2}\right) + c_5(1) + c_6(1) + c_7\left(\frac{n}{2}\right) + c_8\left(\frac{n}{2}\right) + c_9\left(\frac{n}{2}\right) + c_{10}\left(\frac{n}{2}\right)$$

$$= c_1 + c_2 + c_5 + c_6 + \left(\frac{n}{2}\right)(c_3 + c_5 + c_7 + c_8 + c_9 + c_{10})$$

If not found:

$$T(n) = c_1(1) + c_2(1) + c_3\left(\frac{n}{2}\right) + c_7\left(\frac{n}{2}\right) + c_8\left(\frac{n}{2}\right) + c_9\left(\frac{n}{2}\right) + c_{10}\left(\frac{n}{2}\right) + c_{11}(1)$$

$$= c_1 + c_2 + c_{11} + \left(\frac{n}{2}\right)(c_3 + c_5 + c_7 + c_8 + c_9 + c_{10})$$

$$T(n) = \{ \begin{matrix} O(1) & \text{if } n = 1 \\ T\left(\frac{n}{2}\right) + 1 & \text{otherwise} \end{matrix}$$

(seems to be different cases, n = 1 w/ and w/0 it, n =/= 1 w/ and w/o it)

The while-loop will divide the input by 2 at each iteration. This continues until the value is found or not found. Considering that the size of the array is N, and x is the number of iterations, then $\frac{N}{2^x} = 1$. Solving for x, we get $x = \log_2(N)$. This shows that BINARY-SEARCH(A, v) is $O(\log_2(N))$.

3.1-2

Show that for any real constants a and b, where $b > 0$, $(n + a)^b = \Theta(n^b)$.

References:

https://www.csee.umbc.edu/~nam1/TA/HWSols/hw1sol.pdf

$$\Theta(g(n)) = \{f(n)\text{:there exist positive constants } c_1, c_2, \text{and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$$

In this problem, $g(n) = n^b$ and $f(n) = (n + a)^b$.

First the lower bound will be shown:

$$(n + a)^b \leq (n + |a|)^b, \text{where } n > 0$$

$$\leq (n+n)^b, \text{ in cases where } n \geq |a|$$

$$= (2n)^b$$

$$= c_1 n^b, \text{ such that } c_1 = 2^b$$

Then there is the lower bound for $(n+a)^b$,

$$(n+a)^b = \Omega(n^b).$$

To find and upper bound, the following will be shown:

$$(n+a)^b \geq (n-|a|)^b, \text{ where } n > 0$$

$$\geq (c_2 n)^b \text{ where } c_2 = \frac{1}{2} \text{ and } n \geq 2|a|$$

Then there is the upper bound for $(n+a)^b$,

$$(n+a)^b = O(n^b)$$

Then for the values $c_1 = 2^b, c_2 = 2^{-b}$, and $n_0 \geq 2|a|$, there will be $\Theta(n^b)$.

12.1-2

What is the difference between the binary-search-tree property and the min-heap property (p.153)? Can the min-heap property be used to print out the keys of an n-node tree in sorted order in $O(n)$ time? Show how, or explain why not.

References:

http://www.cs.utah.edu/~simpson/classes/cs3510/homework5/homework5.pdf

In a binary-search-tree, each node's key is greater than any key stored in the left subtree, and less than any key stored in its right subtree. In a min-heap, each node's key is less than or equal to the keys of the children.

In a min-heap, it wouldn't make sense to try and print out the keys in sorted order. The min-heap has the structure where the root node is always the smallest value. It would be possible to pop out each value of the array from low to high, but to print out the keys in order would require some other sorting process which would exceed $O(n)$.

12.3-3

We can sort a given set of n numbers by first building a binary search tree containing these numbers (using TREE-INSERT repeatedly to insert the numbers one by one) and then printing the numbers by an inorder tree walk. What are the worst-case and best-case running times for this sorting algorithm?

The best-case and worst-case running times for an inorder tree walk would be $\Theta(n)$ each time. Despite the ordering of the BST, it would require that the tree walk scan through every node one at a time, taking $\Theta(n)$.

21.2-6

Suggest a simple change to the UNION procedure for the linked-list representation that removes the need to keep the *tail* pointer to the last object in each list. Whether or not the weighted-union heuristic is used, your change should not change the asymptotic running time of the UNION procedure. (*Hint:* Rather than appending one list to another, splice them together.)

References:

https://github.com/gzc/CLRS/blob/master/C21-Data-Structures-for-Disjoint-Sets/21.2.md

It is possible to point the linked list members back to their tails for each set. Tracing each set from head to tail, we can find the last element. Combining the set will also have a problem of the last set representative being placed into the other set representative's group, using the smaller one to be included with the bigger one.

Problems for Grading

1. Problem 1 Chapter 2 (Collaborative)

Reference: https://www.cs.odu.edu/~toida/nerzic/content/induction/induction.html

Use induction to prove $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$.

Base case: $n = 1$

In the case of $n = 1$, $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$ becomes to $\sum_{i=1}^{1} i^3 = \left(\frac{1(1+1)}{2}\right)^2$. The LHS $\sum_{i=1}^{1} i^3 = 1^3 = 1$. The RHS $\left(\frac{1(1+1)}{2}\right)^2 = \left(\frac{2}{2}\right)^2 = 1$. So the equation holds true in the base case with $n = 1$.

Induction step: $n = k$

Assume that $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$ holds true in the case of $n = k$. So, the equation $\sum_{i=1}^{k} i^3 = \left(\frac{k(k+1)}{2}\right)^2$ is assumed to be true. (Induction Hypothesis)

To prove for $n = k + 1$, it will be done first for the LHS.

$$\sum_{i=1}^{k+1} i^3 = 1^3 + 2^3 + \cdots + k^3 + (k+1)^3 \tag{1}$$

Using the induction hypothesis, equation (1) can be rewritten as, $\left(\frac{k(k+1)}{2}\right)^2 + (k+1)^3$. This can be further rewritten as,

$$\left(\frac{k(k+1)}{2}\right)^2 + (k+1)^3 = (k+1)^2\left(\frac{k^2}{4} + k + 1\right) = (k+1)^2\left(\frac{k^2+4k+4}{4}\right)$$
$$= \frac{(k+1)^2(k+2)^2}{4} = \left(\frac{(k+1)(k+2)}{2}\right)^2.$$

The last term is equal to the RHS for $n = k+1$. Thus, LHS = RHS for $n = k+1$. ■

2. Problem 2
    a. Use insertion sort to sort the unsorted array < 40, 17, 45, 82, 62, 32, 30, 44, 93, 10 >. Make sure to show the array after every pass.

1. $j = 2$
2. $key = A[2] = 17$
3. $i = 2 - 1 = 1$
4. while $i > 0$ and $A[i] > key$ (executes)
5. $A[2] = A[1]$
    a. < 40, 40, 45, 82, 62, 32, 30, 44, 93, 10 >
6. $i = 1 - 1 = 0$
7. while loop ends
8. $A[1] = key$
    a. < 17, 40, 45, 82, 62, 32, 30, 44, 93, 10 >
9. $j = 3$
10. $key = A[3] = 45$
11. $i = 3 - 1 = 2$
12. while loop skipped
13. $A[3] = key = 45$
    a. < 17, 40, 45, 82, 62, 32, 30, 44, 93, 10 >
14. $j = 4$
15. $key = A[4] = 82$
16. $i = 4 - 1 = 3$
17. while loop skipped
18. $A[4] = key = 82$
    a. < 17, 40, 45, 82, 62, 32, 30, 44, 93, 10 >
19. $j = 5$
20. $key = A[5] = 62$
21. $i = 5 - 1 = 4$
22. while $i > 0$ and $A[i] > key$ (executes)
23. $A[5] = A[4] = 82$
    a. < 17, 40, 45, 82, 82, 32, 30, 44, 93, 10 >

24. $i = 4 - 1 = 3$
25. while loop ends
26. $A[4] = key = 62$
    a. < 17, 40, 45, 62, 82, 32, 30, 44, 93, 10 >
27. $j = 6$
28. $key = A[6] = 32$
29. $i = 6 - 1 = 5$
30. while $i > 0$ and $A[i] > key$ (executes)
31. $A[6] = A[5] = 82$
    a. < 17, 40, 45, 62, 82, 82, 30, 44, 93, 10 >
32. $i = 5 - 1 = 4$
33. while $i > 0$ and $A[i] > key$ (executes)
34. $A[5] = A[4] = 62$
    a. < 17, 40, 45, 62, 62, 82, 30, 44, 93, 10 >
35. $i = 4 - 1 = 3$
36. while $i > 0$ and $A[i] > key$ (executes)
37. $A[4] = A[3] = 45$
    a. < 17, 40, 45, 45, 62, 82, 30, 44, 93, 10 >
38. $i = 3 - 1 = 2$
39. while $i > 0$ and $A[i] > key$ (executes)
40. $A[3] = A[2]$
    a. < 17, 40, 40, 45, 62, 82, 30, 44, 93, 10 >
41. $i = 2 - 1 = 1$
42. while loop ends
43. $A[2] = key = 32$
    a. < 17, 32, 40, 45, 62, 82, 30, 44, 93, 10 >
44. $j = 7$
45. $key = A[7] = 30$
46. $i = 7 - 1 = 6$
47. while $i > 0$ and $A[i] > key$ (executes)
48. $A[7] = A[6] = 82$
    a. < 17, 32, 40, 45, 62, 82, 82, 44, 93, 10 >
49. $i = 6 - 1 = 5$
50. while $i > 0$ and $A[i] > key$ (executes)
51. $A[6] = A[5] = 62$
    a. < 17, 32, 40, 45, 62, 62, 82, 44, 93, 10 >
52. $i = 5 - 1 = 4$
53. while $i > 0$ and $A[i] > key$ (executes)
54. $A[5] = A[4] = 45$
    a. < 17, 32, 40, 45, 45, 62, 82, 44, 93, 10 >
55. $i = 4 - 1 = 3$
56. while $i > 0$ and $A[i] > key$ (executes)

57. $A[4] = A[3] = 40$
    a. < 17, 32, 40, 40, 45, 62, 82, 44, 93, 10 >
58. $i = 3 - 1 = 2$
59. while $i > 0$ and $A[i] > key$ (executes)
60. $A[3] = A[2] = 32$
    a. < 17, 30, 32, 40, 45, 62, 82, 44, 93, 10 >
61. $i = 2 - 1 = 1$
62. while loop ends
63. $A[2] = key = 30$
    a. < 17, 30, 32, 40, 45, 62, 82, 44, 93, 10 >
64. $j = 8$
65. $key = A[8] = 44$
66. $i = 8 - 1 = 7$
67. while $i > 0$ and $A[i] > key$ (executes)
68. $A[8] = A[7] = 82$
    a. < 17, 30, 32, 40, 45, 62, 82, 82, 93, 10 >
69. $i = 7 - 1 = 6$
70. while $i > 0$ and $A[i] > key$ (executes)
71. $A[7] = A[6] = 62$
    a. < 17, 30, 32, 40, 45, 62, 62, 82, 93, 10 >
72. $i = 6 - 1 = 5$
73. while $i > 0$ and $A[i] > key$ (executes)
74. $A[6] = A[5] = 45$
    a. < 17, 30, 32, 40, 45, 45, 62, 82, 93, 10 >
75. $i = 5 - 1 = 4$
76. while loop ends
77. $A[5] = key = 44$
    a. < 17, 30, 32, 40, 44, 45, 62, 82, 93, 10 >
78. $j = 9$
79. $key = A[9] = 93$
80. $i = 9 - 1 = 8$
81. while loop skipped
82. $A[9] = key = 93$
    a. < 17, 30, 32, 40, 44, 45, 62, 82, 93, 10 >
83. $j = A.length = 10$
84. $key = A[10] = 10$
85. $i = 10 - 1 = 9$
86. while $i > 0$ and $A[i] > key$ (executes)
87. $A[10] = A[9] = 93$
    a. < 17, 30, 32, 40, 44, 45, 62, 82, 93, 93 >
88. $i = 9 - 1 = 8$
89. while $i > 0$ and $A[i] > key$ (executes)

90. $A[9] = A[8] = 82$
    a. < 17, 30, 32, 40, 44, 45, 62, 82, 82, 93 >
91. $i = 8 - 1 = 7$
92. while $i > 0$ and $A[i] > key$ (executes)
93. $A[8] = A[7] = 62$
    a. < 17, 30, 32, 40, 44, 45, 62, 62, 82, 93 >
94. $i = 7 - 1 = 6$
95. while $i > 0$ and $A[i] > key$ (executes)
96. $A[7] = A[6] = 45$
    a. < 17, 30, 32, 40, 44, 45, 45, 62, 82, 93 >
97. $i = 6 - 1 = 5$
98. while $i > 0$ and $A[i] > key$ (executes)
99. $A[6] = A[5] = 44$
    a. < 17, 30, 32, 40, 44, 44, 45, 62, 82, 93 >
100. $i = 5 - 1 = 4$
101. while $i > 0$ and $A[i] > key$ (executes)
102. $A[5] = A[4] = 40$
    a. < 17, 30, 32, 40, 40, 44, 45, 62, 82, 93 >
103. $i = 4 - 1 = 3$
104. while $i > 0$ and $A[i] > key$ (executes)
105. $A[4] = A[3] = 32$
    a. < 17, 30, 32, 32, 40, 44, 45, 62, 82, 93 >
106. $i = 3 - 1 = 2$
107. while $i > 0$ and $A[i] > key$ (executes)
108. $A[3] = A[2] = 30$
    a. < 17, 30, 30, 32, 40, 44, 45, 62, 82, 93 >
109. $i = 2 - 1 = 1$
110. while $i > 0$ and $A[i] > key$ (executes)
111. $A[2] = A[1] = 17$
    a. < 17, 17, 30, 32, 40, 44, 45, 62, 82, 93 >
112. $i = 1 - 1 = 0$
113. while loop ends
114. $A[1] = key = 10$
    a. < 10, 17, 30, 32, 40, 44, 45, 62, 82, 93 >

Insertion sort completed.

b. Use merge sort to sort the unsorted array < 75, 56, 85, 90, 49, 26, 12, 48, 40, 47 >. Make sure to show the steps of splitting the array then merging the array.

References:

https://www.google.com/search?q=merge+sort+recursive&oq=merge+sort+recu&aqs=chrome.0.0j69i57j0l4.3832j0j7&sourceid=chrome&ie=UTF-8#kpvalbx=_lqNpXb2TPMHy-gS-voaIAQ18

Using MERGE-SORT(A, p, r) and MERGE(A, p, q, r) (functions are from the textbook)

Let A = < 75, 56, 85, 90, 49, 26, 12, 48, 40, 47 >.

Order of steps:

1. MERGE-SORT(A, 1, A.*length*)
2. $1 < 10$
3. $q = floor\left(\frac{(p+r)}{2}\right) = floor\left(\frac{11}{2}\right) = 5$
4. MERGE-SORT(A, 1, 5)
    a. First split, dividing the array into two halves:
        i. 75, 56, 85, 90, 49
        ii. 26, 12, 48, 40, 47
5. $1 < 5$
6. $q = \cdots = floor\left(\frac{6}{2}\right) = 3$
7. MERGE-SORT(A, 1, 3)
    a. Second split, dividing first half into halves
        i. 75, 56, 85
        ii. 90, 49
8. $1 < 3$
9. $q = \cdots = floor\left(\frac{4}{2}\right) = 2$
10. MERGE-SORT(A, 1, 2)
    a. Third split, dividing into another two halves
        i. 75, 56
        ii. 85
11. $1 < 2$
12. $q = \cdots = floor\left(\frac{3}{2}\right) = 1$
13. MERGE-SORT(A, 1, 1)
14. $1 \nless 1$
15. Exit MERGE-SORT(A, 1, 1)
16. MERGE-SORT(A, 2, 2)
17. $2 \nless 2$
18. Exit MERGE-SORT(A, 2, 2)
19. MERGE(A, 1, 1, 2)
    a. First merge, changing the 75, 56 into
    b. 56, 75
20. A = < 56, 75, 85, 90, 49, 26, 12, 48, 40, 47 > (line 10 exits)
21. MERGE-SORT(A, 3, 3)
22. $3 \nless 3$
23. Exit MERGE-SORT(A, 3, 3)
24. MERGE(A, 1, 2, 3)
    a. Second merge, leaving 56, 75, 85 as is

25. A = < 56, 75, 85, 90, 49, 26, 12, 48, 40, 47 > (line 7 closes)
26. MERGE-SORT(A, 4, 5)
    a. Fourth split, separating 85, 90 from the rest of the first half
27. $4 < 5$
28. $q = \cdots = floor\left(\frac{9}{2}\right) = 4$
29. MERGE-SORT(A, 4, 4)
30. $4 \nless 4$
31. Exit MERGE-SORT(A, 4, 4)
32. MERGE-SORT(A, 5, 5)
33. $5 \nless 5$
34. Exit MERGE-SORT(A, 5, 5)
35. MERGE(A, 4, 4, 5)
    a. Third merge, leaving 85, 90 as is
36. A = < 56, 75, 85, 49, 90, 26, 12, 48, 40, 47 > (line 26 closes)
37. MERGE (A, 1, 3, 5)
    a. Fourth merge, organizing the first half in order
38. A = < 49, 56, 75, 85, 90, 26, 12, 48, 40, 47 > (line 4 closes)
39. MERGE-SORT(A, 6, 10)
    a. Fifth split, separating the original second half
40. $6 < 10$
41. $q = \cdots = floor\left(\frac{16}{2}\right) = 8$
42. MERGE-SORT(A, 6, 8)
    a. Sixth split, separating 26, 12, 48 from the rest of the second half
43. $6 < 8$
44. $q = \cdots = floor\left(\frac{14}{2}\right) = 7$
45. MERGE-SORT(A, 6, 7)
    a. Seventh split, separating 26, 12 from 26, 12, 48
46. $6 < 7$
47. $q = \cdots = floor\left(\frac{13}{2}\right) = 6$
48. MERGE-SORT(A, 6, 6)
49. $6 \nless 6$
50. Exit MERGE-SORT(A, 6, 6)
51. MERGE-SORT(A, 7, 7)
52. $7 \nless 7$
53. Exit MERGE-SORT(A, 7, 7)
54. MERGE(A, 6, 6, 7)
    a. Fifth merge, combining 26, 12 into 12, 26
55. A = < 49, 56, 75, 85, 90, 12, 26, 48, 40, 47 > (line 45 closes)
56. MERGE-SORT(A, 8, 8)
57. Exit MERGE-SORT(A, 8, 8)
58. MERGE(A, 6, 7, 8)

    a. Sixth merge, leaving 12, 26, 48 as is
59. A = < 49, 56, 75, 85, 90, 12, 26, 48, 40, 47 > (line 42 closes)
60. MERGE-SORT(A, 9, 10)
    a. Eighth split, separating 40, 47 from the rest of the second half
61. $9 < 10$
62. $q = \cdots = floor\left(\frac{19}{2}\right) = 9$
63. MERGE-SORT(A, 9, 9)
64. $9 \nless 9$
65. Exit MERGE-SORT(A, 9, 9)
66. MERGE-SORT(A, 10, 10)
67. $10 \nless 10$
68. Exit MERGE-SORT(A, 10, 10)
69. MERGE(A, 9, 9, 10)
    a. Seventh merge, leaving 40, 47 as is
70. A = < 49, 56, 75, 85, 90, 12, 26, 48, 40, 47 > (line 60 closes)
71. MERGE(A, 6, 8, 10)
    a. Eighth merge, organizing the second half to 12, 26, 48, 40, 47
72. A = < 49, 56, 75, 85, 90, 12, 26, 40, 47, 48 > (line 39 closes)
73. MERGE(A, 1, 5, 10)
    a. Final merge, reorganizing both original two halves back into one
74. A = < 12, 26, 40, 47, 48, 49, 56, 75, 85, 90 > (line 1 closes)

c. Show that insertion sort can sort the n/k sublists, each of length k, in Θ(nk) worst-case time.

https://cs.stackexchange.com/questions/52252/how-do-i-analyze-mergesort-that-uses-insertion-sort-for-small-inputs

https://cs.stackexchange.com/questions/68179/combining-merge-sort-and-insertion-sort

The INSERTION-SORT(A) from the textbook will be utilized. It has a worst-case run time of $\Theta(n^2)$. Given that there are $\frac{n}{k}$ sublists each of length k, then the length of the array A is also k. Therefore, in the each of the sublists, the complexity will be $\Theta(k^2)$. Then since there are $\frac{n}{k}$ sublists, each with complexity $\Theta(k^2)$, then the worst-case run time for INSERTION-SORT(A) to sort the sublists is $\Theta\left(\left(\frac{n}{k}\right)k^2\right)$ or $\Theta(nk)$.

d. Show how to merge the sublists in Θ(nlg(n/k)) worst-case time.

To merge the sublists in Θ(nlg(n/k)) requires the use of pairwise merging for the $\frac{n}{k}$ sublists to form a single list. The complexity for the pairwise merging is $\Theta(n)$ at each level of merge, since like regular merge sort we are combining the elements back into n-element lists although they are split into different lists. The total number of levels of going from $\frac{n}{k}$ sublists back into one list is $ceiling\left(\lg\frac{n}{k}\right)$. The merging then takes $n \cdot \lg\frac{n}{k}$ in total, or $\Theta\left(n\lg\frac{n}{k}\right)$.

e. Given that the modified algorithm runs in Θ(nk + nlg(n/k)) worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ-notation?

Standard merge sort has a running time of $\Theta(n\lg n)$. Then we must find a value of k that would allow $\Theta(n\lg n) = \Theta\left(nk + n\lg\frac{n}{k}\right)$. The maximum value of k is $\Theta(\lg n)$, otherwise there would be a higher order term compared to $n\lg n$ (which would make it of a longer run-time). Plugging in this value into $\Theta\left(nk + n\lg\frac{n}{k}\right)$ leads to:

$$n\lg n + n\lg\frac{n}{\lg n} = n\lg n + n\lg n - n\lg\lg n$$

$$= 2n\lg n - n\lg\lg n.$$

Looking at $\Theta(2n\lg n - n\lg\lg n)$, we can ignore $-n\lg\lg n$ because it is a lower-order term, and we can drop the constant 2. This gives us $\Theta(n\lg n)$.

f. How should we choose k in practice?

To determine which k is ideal would imply that it is a value where the length of each sublist $\frac{n}{k}$ is as large as possible, while insertion sort is still faster than using ordinary merge sort. To determine such a value requires machine computation to determine.

3. Problem 3. Write a $\Theta(m + n)$ algorithm that prints the in-degree and the out-degree of every vertex in an m-edge, n-vertex directed graph where the directed graph is represented using adjacency lists.

References:

https://brilliant.org/wiki/depth-first-search-dfs/#complexity-of-depth-first-search

http://www.bowdoin.edu/~ltoma/teaching/cs231/fall09/Homeworks/old/rest/H10-sol.pdf

The following is a depth first search algorithm modified from the textbook, p.604. The algorithm uses similar attributes as the ones in the book such as 'color' and 'π'. Additional attributes 'in' and 'out' have been added which each represent the in-degree and out-degree of each vertex. Attributes related to the 'time' variable have been removed since they are not necessary.

DFS(G)

```
1. for each vertex $u \in G.V$
2.     $u.color = WHITE$
3.     $u.\pi = NIL$
4.     $u.out = 0$
```

```
5.   $u.in = 0$
6. for each vertex $u \in G.V$
7.   if $u.color == WHITE$
8.     DFS-VISIT$(G, u)$
9.   print "In-degree of " $u$ " is " $u.in$
```

DFS-VISIT(G, u)

```
1. $u.color = GRAY$
2. for each $v \in G.Adj[u]$
3.   $u.out = u.out + 1$
4.   $v.in = v.in + 1$
5.   if $v.color == WHITE$
6.     $v.\pi = u$
7.     DFS-VISIT$(G, v)$
8. print "Out-degree of " $u$ " is: " $u.out$
9. $u.color = BLACK$
```

4. Problem 4 Ch. 12 BST

Exercise 12.2-1 Suppose that we have numbers between 1 and 1000 in a binary search tree and we want to search for the number 363. Which of the following sequences could not be the sequence of nodes examined?

i. 2, 252, 401, 398, 330, 397, 363.

2 < 252, 401, 398, 330, 397, 363

252 < 401, 398, 330, 397, 363

401 > 398, 330, 397, 363

398 > 330, 397, 363

330 < 397, 363

397 > 363

ii. 924, 220, 911, 244, 898, 258, 362, 363.

924 > 220, 911, 244, 898, 258, 362, 363

220 < 911, 244, 898, 258, 362, 363

911 > 244, 898, 258, 362, 363

244 < 898, 258, 362, 363

898 > 258, 362, 363

258 < 362, 363

362 < 363

iii. 925, 202, 911, 240, 912, 245, 363.

925 > 202, 911, 240, 912, 245, 363

202 < 911, 240, 912, 245, 363

911 ≯ 240, 912, 245, 363; since 911 < 912. Therefore, this sequence is not one of the possible sequences examined.

iv. 2, 399, 387, 219, 266, 382, 381, 278, 363.

2 < 399, 387, 219, 266, 382, 381, 278, 363

399 > 387, 219, 266, 382, 381, 278, 363

387 > 219, 266, 382, 381, 278, 363

219 < 266, 382, 381, 278, 363

266 < 382, 381, 278, 363

382 > 381, 278, 363

381 > 278, 363

278 < 363

v. 935, 278, 347, 621, 299, 392, 358, 363.

935 > 278, 347, 621, 299, 392, 358, 363

278 < 347, 621, 299, 392, 358, 363

347 ≮ 621, 299, 392, 358, 363; since 347 > 299. Therefore, this sequence is not one of the possible sequences examined.

5. Problem 5 Sorting Iris Plants (Collaborative)

Develop an algorithm to sort the five features in the dataset to determine if any of the five sorted features can separate the three plant types. Show the efficiency of your sorting algorithm based on its ability to sort the five sets of features.

In this problem the algorithm used was selection sort. The result from sorting the features individually showed that for Sepal Length and Sepal Width, there is still considerable overlap between the classes. However, when sorting Petal Length and Petal Width, the classes remain quite well separated. There is still some slight overlap for classes virginica and versicolor, but it is not that great compared to the overlap when sorting the first two features (i.e., sepal length and sepal width). Below is the R code used for the problem.

```r
# Selection sort modified to return the sorted indices
selection_sort <- function(rand_array) {
  idx_vec = 1:length(rand_array)
  for (j in 1:(length(rand_array) - 1)) {  # Loops through 1-n
    min_key = rand_array[j]  # creates a default min key
    min_key_idx = idx_vec[j]
    for (i in (j+1):length(rand_array)) {  # Loops through remaining array el
ements
      # select min value from remaining array
      if (rand_array[i] < min_key) {
        # swap min key and rand_array[i]
        temp = min_key
        min_key = rand_array[i]
        rand_array[i] = temp

        temp_idx = min_key_idx
        min_key_idx = idx_vec[i]
        idx_vec[i] = temp_idx
      }
    }
    rand_array[j] = min_key  # save min key to next index of array
    idx_vec[j] = min_key_idx
  }
  return(idx_vec)
}


# Copy over iris data
iris_data <- iris

# Save row index to separate column
iris_data['idx'] <- rownames(iris_data)

# Sort dataframe by each feature
feature_1 <- iris_data[selection_sort(iris_data$Sepal.Length),]
feature_2 <- iris_data[selection_sort(iris_data$Sepal.Width),]
feature_3 <- iris_data[selection_sort(iris_data$Petal.Length),]
feature_4 <- iris_data[selection_sort(iris_data$Petal.Width),]

# Reset row indices
rownames(feature_1) <- NULL
rownames(feature_2) <- NULL
rownames(feature_3) <- NULL
rownames(feature_4) <- NULL
```

```r
plot((1:nrow(feature_1)), feature_1$Species)
plot((1:nrow(feature_2)), feature_2$Species)
plot((1:nrow(feature_3)), feature_3$Species)
plot((1:nrow(feature_4)), feature_4$Species)
```