Exam 1

1. Which Big-O expression best characterizes the worst time complexity of the following code?

3 public static int foo(int N) {  
 4 int count = 0;  
 5 for (int i = 0; i < 10; i = i + 2) {  
 6 for (int j = i; j < 20; j = j + 1) {  
 7 int k = 1;  
 8 while (k < 30) {  
 9 count++;  
10 k = k + 1;  
11 }  
12 }  
13 }  
14 return count;  
15 }

Answer: A - O(1)

1. Which Big-O expression best characterizes the worst time complexity of the following code?

21 public static int foo(int N) {  
22 int count = 0;  
23 for (int i = 0; i < N; i = i + 2) {  
24 for (int j = i; j < N; j = j + 1) {  
25 int k = 1;  
26 while (k < N / 2) {  
27 count++;  
28 k = k + 1;  
29 }  
30 }  
31 }  
32 return count;  
33 }

Answer: D – O(N^3)

1. Which Big-O expression best characterizes the worst time complexity of the following code?

35 public static int foo(int N) {  
36 int count = 0;  
37 int i = N;  
38 while (i > 1) {  
39 count++;  
40 i = i / 2;  
41 }  
42 for (int j = 1; j < N; j++) {  
43 count++;  
44 }  
45 return count;  
46 }

Answer: B – O(N)

1. Which Big-O expression best characterizes the worst time complexity of the following code?

50 public static int foo(int N) {  
51 int count = 0;  
52 int i = N;  
53 while (i > 1) {  
54 count++;  
55 i = i / 2;  
56 }  
57 for (int j = 1; j < 10; j++) {  
58 count++;  
59 }  
60 return count;  
61 }

Answer: A – O(logN)

1. Which Big-O expression best characterizes the worst time complexity of the following code?

65 public static int foo(int N) {  
66 int count = 0;  
67 int i = 1;  
68 while (i < N) {  
69 for (int j = 1; j < N; j = j + 2) {  
70 count++;  
71 }  
72 i = i \* 2;  
73 }  
74 return count;  
75 }

Answer: C – O(N log N)

1. Which Big-O expression best characterizes the worst time complexity of the following code?

79 public static int foo(int N) {  
80 int count = 0;  
81 int i = 1;  
82 while (i < N) {  
83 for (int j = 1; j < N; j = j + 2) {  
84 count++;  
85 }  
86 i = i + 2;  
87 }  
88 return count;  
89 }

Answer: D – O(N^2)

1. Suppose you attempted to empirically discover the big O running time of a program and you were able to generate the following timing data.

|  |  |  |
| --- | --- | --- |
| N | Time | Ratio |
| 128 | 0.622 | - |
| 256 | 8.886 | 14.286 |
| 512 | 141.048 | 15.873 |
| 1024 | 2249.704 | 15.950 |
| 2048 | 35958.996 | 15.984 |

In the table above, the N column records the size of the input for each run, the Time column records the elapsed time in seconds for each run, and Ratio is the elapsed time for the current run divided by the lapsed time the previous room. (i.e. Time1/Time i-1).

Based on the timing data presented in this table, what is the most reasonable conclusion regarding the underlying big O time complexity of the program being timed?

Answer: D – O(N^4)

1. Suppose you have implemented an algorithm in a method named foo, which takes an array of N floating-point numbers as data. Suppose also that this algorithm has O(N^3) best, average, and worst-case time complexity and that a timing analysis of foo showed that approximately 2 seconds were required to process an array of size N = 256.

What is the largest array (N) that foo could process in less than one hour?

Answer: C – 2048

1. Assuming that the parameter a contains unique values, what does method foo do?

93 public static int foo(int[] a, int k) {  
 94 for (int i = 0; i < k; i++) {  
 95 int location = i;  
 96 for (int j = i + 1; j < a.length; j++) {  
 97 if (a[j] < a[location]) {  
 98 location = j;  
 99 }  
100 }  
101 swap(a, i, location);  
102 }  
103 return a[k - 1];  
104 }

Answer: B – returns the kth smallest value in a

1. What is the maximum number of elements that would have to be examined on any binary search of the array a below?

A = [3, 5, 7, 13, 15, 17, 19, 21, 25, 33, 37, 41, 51, 72, 88, 91]

(16 elements)

Answer: B – 5

1. The max method below is intended to return the largest value in the parameter *coll.* which test case parameter would expose the logic error in this method?

109 public <T extends Comparable<T>> T max(Collection<T> coll) {  
110 Iterator<T> itr = coll.iterator();  
111 T max = itr.next();  
112 while (itr.hasNext()) {  
113 if (max.compareTo(itr.next()) < 0) {  
114 max = itr.next();  
115 }  
116 }  
117 return max;  
118 }

Answer: A – coll = [10, 20, 15, 5, 7]

1. The min method below is intended to return the smallest of its three int parameters. Which call below would expose the logic error in this method?

122 public static int min(int a, int b, int c) {  
123 if ((a < b) && (a < c)) {  
124 return a;  
125 }  
126 if ((b < a) && (b < c)) {  
127 return b;  
128 }  
129 return c;  
130 }

Answer: C – min(5, 5, 8)

1. What would happen if you tried to compile and run the following code?

134 public class MyClass {  
135   
136 public static void main(String[] args) {  
137 Comparable[] a = {"hi", new Double(1.14), new Integer(3)};  
138 Comparable key = new Float(3.14);  
139 System.out.println(seach(a, key));  
140 }  
141   
142 public static boolean search(Comparable[] a, Comparable key) {  
143 for (Comparable value : a) {  
144 if (value.compareTo(key) == 0) {  
145 return true;  
146 }  
147 }  
148 return true;  
149 }  
150 }

Answer: B – Compilation succeeds but a runtime error is generated.

1. Which method supports type-safety and would compile with no warnings or errors?

153 //A  
154   
155 public <T extends Comparable<T>> boolean search(Collection<T> coll, T target) {  
156 for (T element : coll) {  
157 if (element.compareTo(target) == 0) {  
158 return true;  
159 }  
160 }  
161 return false;  
162 }  
163   
164 //B  
165   
166 public <T extends Comparable> boolean search(Collection<T> coll, T target) {  
167 for (T element : coll) {  
168 if (element.compareTo(target) == 0) {  
169 return true;  
170 }  
171 }  
172 return false;  
173 }  
174   
175 //C  
176   
177 public boolean search(Collection<Comparable> coll, Comparable target) {  
178 for (Comparable element : coll) {  
179 if (element.compareTo(target) == 0) {  
180 return true;  
181 }  
182 }  
183 return false;  
184 }  
185   
186 //D  
187   
188 public <T extends Comparable> boolean search(Collection<Comparable> coll, T target) {  
189 for (Comparable element : coll) {  
190 if (element.compareTo(target) == 0) {  
191 return true;  
192 }  
193 }  
194 return false;  
195 }

Answer: A – T extends Comparable<T>

1. What recursive call should fill the blank in the method below to correctly compute the sum of the integers from 1 to n?

199 public int sum(int n) {  
200 if (n == 1) {  
201 return 1;  
202 }  
203 return n + \_\_\_\_\_\_\_\_\_\_\_\_;  
204 }  
205

Answer: D – sum(n – 1)

1. Which of the following algorithms was not an in place sort?

Answer: C – merge sort

1. Suppose you have an array full of instances of a class named Person. Each person has a name and a date of birth. The following is an example of such an array.

[Joe 02-21-97, Amy 06-10-95, Jan 04-07-96, Bob 11-30-95, Pat 02-21-97, Lee 04-07-96]

To arrange the elements in this array in ascending date of birth order, and where multiple persons have the same date of birth they are listed in ascending order of name, which is the correct sequence of sorts to perform? (An example od the desired order is shown below.)

[Joe 02-21-97, Pat 02-21-97, Jan 04-07-96, Lee 04-07-96, Amy 06-10-95, Bob 11-30-95]

1. Use merge sort to sort the array in ascending order of name, then use merge sort to sort the array in ascending order of date of birth.
2. Given the array a = [66, 67, 20, 86, 55, 74, 11, 91, 43, 47] which sorting algorithm, would perform the following sequence of array modifications?

[66, 67, 20, 86, 55, 74, 11, 91, 43, 47]

[20, 66, 67, 86, 55, 74, 11, 91, 43, 47]

[20, 66, 67, 86, 55, 74, 11, 91, 43, 47]

[20, 55, 66, 67, 86, 74, 11, 91, 43, 47]

[20, 55, 66, 67, 74, 86, 11, 91, 43, 47]

[11, 20, 55, 66, 67, 74, 86, 91, 43, 47]

[11, 20, 55, 66, 67, 74, 86, 91, 43, 47]

[11, 20, 43, 55, 66, 67, 74, 86, 91, 47]

[11, 20, 43, 47, 55, 66, 67, 74, 86, 91]

Answer: b – insertion sort

1. Given the array a = [66, 67, 20, 86, 55, 74, 11, 91, 43, 47] which sorting algorithm, would perform the following sequence of array modifications?

[66, 67, 20, 86, 55, 74, 11, 91, 43, 47]

[20, 66, 67, 86, 55, 74, 11, 91, 43, 47]

[20, 66, 67, 55, 86, 74, 11, 91, 43, 47]

[20, 55, 66, 67, 86, 74, 11, 91, 43, 47]

[20, 55, 66, 67, 86, 11, 74, 91, 43, 47]

[20, 55, 66, 67, 86, 11, 74, 91, 43, 47]

[20, 55, 66, 67, 86, 11, 43, 47, 74, 91]

[11, 20, 43, 47, 55, 66, 67, 74, 86, 91]

Answer: C – merge sort

1. Which of the arrays below would be the final result of partitioning the following portion of an array using 59 as the pivot in the quicksort partition implementation presented in lecture? Only the partitioning operation is happening.

[97, 20, 84, 24, 25, 59, 93, 13, 94]

Answer: C – [20, 24, 25, 13, 59, 94, 93, 97, 84]

1. Given the array a = [66, 67, 20, 86, 55, 74, 11, 91, 43, 47] which sorting algorithm, would perform the following sequence of array modifications?

[11, 67, 20, 86, 55, 74, 66, 91, 43, 47]

[11, 20, 67, 86, 55, 74, 66, 91, 43, 47]

[11, 20, 43, 86, 55, 74, 66, 91, 67, 47]

[11, 20, 43, 47, 55, 74, 66, 91, 67, 86]

[11, 20, 43, 47, 55, 74, 66, 91, 67, 86]

[11, 20, 43, 47, 55, 66, 74, 91, 67, 86]

[11, 20, 43, 47, 55, 66, 67, 91, 74, 86]

[11, 20, 43, 47, 55, 66, 67, 74, 91, 86]

[11, 20, 43, 47, 55, 66, 67, 74, 86, 91]

Answer: A – selection sort

1. Suppose you have collected data from several timing experiments and you have characterized a sorting method’s time complexity profile as follows.
   1. Data in ascending order: O(N)
   2. Data in descending order: O(N^2)
   3. Data in random order: O(N^2)

Which sorting algorithm is implemented in this method?

Answer: B – insertion sort

1. Suppose you have collected data from several timing experiments and you have characterized a sorting method’s time complexity profile as follows.
   1. Data in ascending order: O(N log N)
   2. Data in descending order: O(N log N)
   3. Data in random order: O(N log N)

Which sorting algorithm is implemented in this method?

Answer: C – merge sort

1. Suppose you have collected data from several timing experiments and you have characterized a sorting method’s time complexity profile as follows.
   1. Data in ascending order: O(N^2)
   2. Data in descending order: O(N^2)
   3. Data in random order: O(N log N)

Which sorting algorithm is implemented in this method?

Answer: D – quicksort (without randomization)

1. Suppose you have collected data from several timing experiments and you have characterized a sorting method’s time complexity profile as follows.
   1. Data in ascending order: O(N^2)
   2. Data in descending order: O(N^2)
   3. Data in random order: O(N^2)

Which sorting algorithm is implemented in this method?

Answer: A – selection sort

Exam 2

1. Consider the RandomizedList interface shown below.

210 public interface RandomizedList extends List {  
211   
212 void add(T element);  
213   
214 T remove();  
215   
216 T sample();  
217   
218 }

Suppose that the following performance guarantees are make for a particular implementing class for this interface.

Add: O(1)

Remove: O(1)

Sample: O(1)

Which data structure is being used in this implementing class?

Answer: B – an array with values maintained in no particular order

1. The numbers in the two-dimensional array below indicate the order in which each position in the array was examined by a search algorithm. Which search algorithm does this sequence suggest most strongly?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 26 | 27 | 28 | 29 | 30 |
| 17 | 18 | 19 | 22 | 23 |
| 20 | 10 | 11 | 12 | 15 |
| 21 | 13 | 2 | 3 | 4 |
| 24 | 14 | 5 | 1 | 6 |
| 25 | 16 | 7 | 8 | 9 |

Answer: C – breadth first search

1. The numbers in the two-dimensional array below indicate the order in which each position in the array was examined by a search algorithm. Which search algorithm does this sequence suggest most strongly?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 2 | 1 | 9 |
| 5 | 6 | 7 | 8 | 10 |
| 14 | 13 | 12 | 11 | 19 |
| 15 | 16 | 17 | 18 | 20 |
| 24 | 23 | 22 | 21 | 29 |
| 25 | 26 | 27 | 28 | 30 |

Answer: A – depth first search

1. Recall that an array-based queue based on the “circular array” strategy does not anchor its front at the 0 index but instead keeps track of its front position and allows it to change. Which method below has O(1) time complexity in a “circular array” implementation but has O(N) time complexity in a “left justified, anchored at 0 array” implementation?

Answer: B – dequeue / remove

1. Suppose the object q below is an instance of a class that uses a “circular array” scheme to implement a queue. Assume that the front and rear markers begin at index 0. Suppose also that the queue has a fixed capacity of 5. That is, there is no array resizing done.

Which choice below depicts the contents of the internal array after the following sequence of operations? (The array is shown from left to right beginning at index 0. Thee symbol \* is used to denote an empty cell.)

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

q.dequeue();

q.enqueue(4);

q.enqueue(5);

q.dequeue();

q.enqueue(6);

q.dequeue();

q.enqueue(7);

Answer: D – [6, 7, \*, 4, 5]

1. Recall the stack-based “shunting yard algorithm” from class for evaluating postfix expressions. If it were applied to the following postfix expression, what would the stack contain after the division operator (/) has been processed but before the subsequent subtraction operator (-) has been read?

2 5 3 – 4 1 + X 8 2 / - x

Answer: A. top| 4, 10, 2

1. What does the queue q contain after the following sequence of operations? Note that the front of queue q is the left-most element listed and the rear of queue q is the right-most element listed.

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

q.enqueue(4);

q.dequeue();

q.dequeue();

q.enqueue(5);

q.enqueue(6);

q.dequeue();

Answer: C – front | 4, 5, 6 | rear

1. What does the stack s contain after the following sequence of operations? Note that the top of stack s is the left-most element listed.

s.push(1);

s.push(2);

s.push(3);

s.push(4);

s.pop();

s.push(5);

s.push(6);

s.pop();

s.pop();

s.pop();

s.push(7);

Answer: B – top | 7, 2, 1

1. What doubly-linked list of nodes is accessible from n after the following statements have executed?

Node n = new Node(1);

Node m = new Node(2);

Node p = new Node(3);

n.prev = null;

m.next = null;

n.next = p;

p.next = m;

m.prev = p;

p.prev = n;

p = null;

m = null;

Answer: D – n > [1] <> [3] <> [2]

1. Supposed n, m, and p are references to Node objects as shown in the image below. Which set of statements would modify the pointer chain so that the nodes accessible from n are in the order A, B, E, D? (That is, the pointer chain referenced from n becomes [A] > [B] > [E] > [D].)

n > [A] > [B] > [C] > [E] > null

p^

m > [D] > null

Answer: C – p.next = p.next.next;

p.next.next = m;

1. How many singly-linked Node objects could the Java Virtual Machine mark as “garbage” after all the following statements have executed?

Node n = null;

for (int i = 0; i < 5; i++) {

n = new Node(i, n);

}

Node m = n.next.next.next;

n = null;

Answer: C – 3.

1. What singly-linked list of nodes is accessible from n after the following statements have executed?

Node n = new Node(“A”);

n.next = new Node(“B”, new Node(“C”));

n.next = n.next.neext;

n = new Node(“D”, n);

n.next.next = new Node(“E”);

n.next = new Node(“F”, n.next);

Answer: B – [D] > [F] > [A] > [E]

1. Removing an element from an ordered collection (like an indexed list) can be thought of as a sequential two-part process: (1) Locate the element, then (2) physically remove the element. If a chain of linked nodes is used as the underlying data structure for the collection, the location of the element is identified by a reference to a node in the chain. Assuming the reference to the node containing the element has already been found, what is the worst-case time complexity of then physically removing this node from the chain?

Answer: A. O(1)

1. Removing a element from an ordered collection (like an indexed list) can be thought of as a sequential two-part process: (1) Locate the element, then (2) physically remove the element. If an array is used as the underlying data structure for the collection, the location of the element is identified by a legal index value in the array. Assuming the index of the element has already been found, what is the worst-case time complexity of then physically removing the element from the array?

Answer: C – O(N)

1. Consider the add method of an array-based collection below.

224 public boolean add(T element) {  
225 if (size == elements.length) {  
226 T[] newArray = (T[] new Object[newSize];  
227 for (int i = 0; i < size; i++) {  
228 newArray[i] = elements[i];  
229 }  
230 elements = newArray;  
231 }  
232 elements[size] = element;  
233 size++;  
234 return true;  
235 }

Using the amortized analysis we would say that this method has what time complexity?

Answer: D – O(1)

1. Consider the object b below, an instance of the **ArrayBag** class discussed in lecture and illustrated in lab. Recall that the **ArrayBag** uses an array as the physical storage structure and uses the dynamic resizing strategy exactly as we discussed in class. Assuming the array begins with capacity 1, what will the **capacity** (i.e. length) of the array be after the following sequence of statements are executed?

240 ArrayBag<Integer> b = new ArrayBag<Integer>();  
241 for (int i = 1; i <= 12; i++) {  
242 b.add(i);  
243 }  
244 for (int i = 1; i <= 11; 1++) {  
245 b.remove(i);  
246 }

Answer: A – 4

1. Consider the three methods below on an **IndexedList** collection; that is, a collection in which elements are stored in order by index.

Public void add(Object element, int index)

Public Object remove(int index)

Public Boolean contains(Object element)

Supposed we implemented this collection using an **array** for the physical storage of the elements. Based on the ideas and techniques are have discussed to this point in class, which of the worst-case time complexity profiles below characterize the most efficient implementation strategy that you could guarantee? Assume amortization is included as appropriate to account for array resizing in those marked O(1).

Answer: D – add O(N), remove O(N), contains O(N)

1. Consider the three methods below on an **IndexedList** collection; that is, a collection in which elements are stored in order by index.

Public void add(Object element, int index);

Public Object remove(int index)

Public Boolean contains(Object element)

Suppose we implemented this collection using **linked nodes** for the physical storage of the elements. Based on the ideas and techniques we have discussed to this point in class, which of the worst-case time complexity profiles below characterize the most efficient implementation strategy that you could guarantee?

Answer: B – add O(N), remove O(N), contains O(N)

1. Consider the three methods below on a **Bag** collection; that is a collection in which duplicates are allowed and the values are not required to be kept in any particular order.

Public void add(Comparable value)

Public void remove(Comparable value)

Public void contains(Comparable value)

Suppose we implement this collection using an **array** for the physical storage of the elements, and suppose that we maintain the array values in **ascending order**. Based on the ideas and techniques we have discussed to this point in calss, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee? Assume amortization is included as appropriate to account for array resizing in those marked O(1).

Answer C – Add O(N), remove O(N), contains O(logN)

1. Consider the three methods below on a **Bag** collection; that is, a collection in which duplicates are allowed and the values are not required to be kept in any particular order.

Public void add(Comparable value)

Public void remove(comparable value)

Public void contains(Comparable value)

Suppose we implement this collection using an **array** for the physical storage of the elements, and suppose that we maintain the array values in **no specific order**. Based on the ideas and techniques are have discussed to this point in class, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee? Assume amortization is included as appropriate to account for array resizing in those market O(1).

Answer A – add O(1), remove O(N), contains O(N)

1. Consider the three methods below on a **Set** collection; that is, a collection in which no duplicates are allowed and the values are not required to be kept in any particular order.

Public void add(Comparable value)

Public void remove(Comparable value)

Public void contains(Comparable value)

Suppose we implement this collection using a **singly-linked list** for the physical storage of the elements, and suppose that we maintain the array values in **ascending order**. Based on the ideas and techniques are have discussed to this point in class, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee?

Answer D – add O(N), remove O(N), contains O(N)

1. Consider the three methods below on a **Set** collection; that is, a collection in which no duplicates are allowed and the values are not required to be kept in any particular order.

Public void add(Comparable value)

Public void remove(Comparable value)

Public void contains(Comparable value)

Suppose we implement this collection using a **singly-linked list** for the physical storage of the elements, and suppose that we maintain the array values in **no particular order**. Based on the ideas and techniques are have discussed to this point in class, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee?

Answer D – add O(N), remove O(N), contains O(N)

1. Consider the three methods below on a **Set** collection; that is, a collection in which no duplicates are allowed and the values are not required to be kept in any particular order.

Public void add(Comparable value)

Public void remove(Comparable value)

Public void contains(Comparable value)

Suppose we implement this collection using an **array** for the physical storage of the elements, and suppose that we maintain the array values in **ascending order**. Based on the ideas and techniques are have discussed to this point in class, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee?

Answer C – add O(N), remove O(N), contains O(logN)

1. Consider the three methods below on a **Set** collection; that is, a collection in which no duplicates are allowed and the values are not required to be kept in any particular order.

Public void add(Comparable value)

Public void remove(Comparable value)

Public void contains(Comparable value)

Suppose we implement this collection using an **array** for the physical storage of the elements, and suppose that we maintain the array values in **no specific order**. Based on the ideas and techniques are have discussed to this point in class, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee?

Answer D – add O(N), remove O(N), contains O(N)

1. The code below is similar to something you had to implement in the Collinear Points assignment, Assuming that the comment “identify collinear runs” represents a code segment that itself is O(1) and N represents the number of point sin the array points, what is the overall time complexity of the code shown.

Points[] pointsBySlope = Arrays.<Point>copyOf(points, points.length);

Arrays.<Point>sort(points);

For (int refPoint = 0; refPoint < points.length; refPoint++) {

Arrays.<Point>sort(pointsBySlope, points[refPoint].slopeOrder);

Int start = 0;

Int end = 0;

Int current = 0;

While (current < pointsBySlope.length – 1) {

//identify collinear runs

}

}

Answer B – O(N^2 log N)

Exam 3

1. Which tree would result from inserting the following values in the order in which they are written into an initially empty 2-4 tree? 5, 8, 2, 9, 7, 6, 3, 4, 1

Answer C

3|5|8

1|2 4 6|7 9

1. Which tree would result from inserting the following values in the order in which they are written into an initially empty AVL tree? 5, 4, 6, 7, 8, 3, 2, 1

Answer D

5

3 7

2 4 6 8

1

1. What is the balance factor of the root of the AVL tree below?

12

5 17

2 8 14 21

1 4 7 10 13 15 19 24

3 6 9 11 16 18 20 22 25

23

Height left = 4, Height right = 5

H(R) – H(L) = 1

Answer C – 1

1. Which tree would result from deleting 21 from the AVL tree below?

17

7 22

3 11 19 24

2 5 9 14 18 20 23 25

1 4 6 8 10 12 16 21

13 15

Answer: B –

11

7 17

3 9 14 22

2 5 8 10 12 16 19 24

1. What is the height of a complete binary tree that contains 8 nodes?

Answer: B – 4

1. How many leaves are in a full binary tree of height 5?

Answer: B – 16

1. What is the height of node A in the tree below?

Answer B – 1 (leaves are still height 1)

1. In what order would be nodes of the binary tree below be visited during an inorder traversal?

J

F L

E H K M

G I N

Answer: A – E, F, G, H, I, J, K, L, M, N

1. In what order would the nodes of the binary tree below be visited during a level-order traversal?

N

J P

J L O Q

H K M

Answer: D – N, J, P, I, L, O, Q, H, K, M

1. In what order would be nodes of the binary tree below be visited during a postorder traversal?

O

M S

L N Q T

K P R

Answer: A – K, L, N, M, P, R, Q, T, S, O

1. In what order would be nodes of the binary tree below be visited during a preorder traversal?

R

O T

N P S V

Q U W

Answer: C – R, O, N, P, Q, T, S, V, U, W

1. Which tree would result from inserting the following values in the order in which they are written into an initially empty red-black tree? 5, 4, 6, 7, 8, 3, 2, 1

Answer: A

A drawing of a person

Description automatically generated

1. Given the array a = [15, 17, 11, 13, 12, 18, 16, 14, 20, 19], which response below depicts a after the first phase of the heapsort algorithm as discussed in class, when its elements have been arranged in partial order?

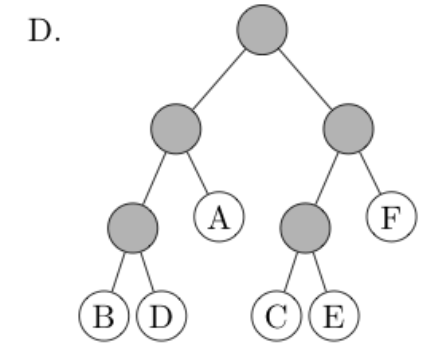
Answer: B

A picture containing object, clock

Description automatically generated

1. Which code tree would Huffman’s algorithm construct for a text file that contains 6 A’s, 2 B’s, 3 C’s, 3 D’s, 4 E’s, and 9 F’s?

Answer: D



1. What is the height of the binary search tree (with no balance constraints) that results from adding the following values in the order in which they are written? 7, 6, 5, 4, 2, 1, 3

Answer: C – 6

1. Assume you have an open-addressed hash table using linear probing for collision resolution and no rehashing. Which table below shows the result of inserting the following hash codes in the order in which they appear, using the hash function h(hashcode) = hashcode % 10?

22, 5, 9, 18, 14, 28, 30, 19

Answer: A

A picture containing object

Description automatically generated

1. Assume you have an open-addressed hash table using double hashing for collision resolution and no rehashing. Which table below shows the result of inserting the following hash codes in the order in which they appear, using the hash function h(hashcode) = hashcode % 10 and the secondary hash function h2(hashcode) = 1 + (hashcode % 9).

22, 5, 9, 18, 14, 28, 30, 19

Answer: D

A picture containing clock

Description automatically generated

1. Consider an open-addressed table with capacity 1200 using double hashing for collision resolution. To guarantee that no more than 4 probes would be needed to satisfy any add, remove, or search request on the table, what is the minimum number of empty buckets that must exist?

Answer: B – 300

1. Which of the following is a valid breadth-first traversal of the graph below?

A picture containing object, clock

Description automatically generated

Answer: C – f, g, j, h, I, k

1. Which of the following is a valid depth-first traversal of the graph below?

A picture containing clock, object

Description automatically generated

Answer: D – k, j, g, f, I, h

1. What would the cost array contain immediately after Dijkstra’s algorithm discovers with certainty the least-cost path from f to I and after this path is used to update the cost estimates of the neighbors of I in the graph shown below?

A picture containing object, watch

Description automatically generated

Answer: A

A close up of a clock

Description automatically generated

1. In what order would Kruskal’s algorithm add the edges to the minimum spanning tree of the graph shown below?

A clock in the middle of a watch

Description automatically generated

Answer: D – 1, 3, 4, 6, 7, 9, 10

1. In what order would Prim’s algorithm add the edges to the minimum spanning tree of the graph shown below if the starting vertex is u?

A clock in the middle of a watch

Description automatically generated

Answer B – 6, 7, 9, 10, 1, 3, 4

1. Select the vertex listing that reflects a topological sort (ordering) of the graph shown below.

A picture containing clock, object

Description automatically generated

Answer D – z, u, w, v, y, z

1. The method below is very similar to one you used for an assignment. What is its time complexity in terms of N, where N is the number of lines in the InputStream?

Public Doublets(InputStream in) {

Try {

Lexicon = new TreeSet<String>();

Scanner s = new Scanner(new BufferedReader(new InputStreamReader(in)

While (s.hasNext()) {

String str = s.next();

Lexicon.add(str);

s.nextLine();

}

In.close();

}

Catch (java.io.IOException e) {

System.err.println(“Error reading from InputStream.”);

System.exit(1);

}

}

Answer: B – N

Practice Exam Module 7

1. The array shown below is the component id field of a Disjoint Set object. Assuming that this disjoint set object uses the *fast find* implementation strategy, how many connected components are represented?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 8 | 2 | 3 | 5 | 5 | 6 | 7 | 8 | 5 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Answer: B – 6

1. Disjoint set – *fast union* . How many connected components?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | 8 | 2 | 3 | 5 | 5 | 6 | 7 | 8 | 5 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Answer: B – 6

1. Disjoint set – *fast union* What is the result of find(6)?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 8 | 3 | 0 | 5 | 1 | 8 | 8 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Answer: C – 1

1. Disjoint set – fast union. What would be the state of the array after union(2, 4)?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 8 | 3 | 0 | 5 | 1 | 8 | 8 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Answer, D:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 8 | 1 | 8 | 3 | 0 | 5 | 1 | 8 | 8 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

1. Assuming that the fast union strategy was used, what is the result of find(6)?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 8 | 1 | 8 | 3 | 0 | 5 | 1 | 8 | 8 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Answer, A

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 8 | 1 | 8 | 3 | 8 | 8 | 1 | 8 | 8 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Random study materials

Properties of sorts

Comparison sort: the only assumption about the data being sorted is that the data elements can be compared to each other on the basis of “less than” – that is, a total order exists.

In-place: The list itself is rearranged and only a constant amount of extra space is required.

Adaptive: Running time can be improved by taking advantage of certain input arrangements.

Stable: Equal elements maintain the same relative order.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Selection | Insertion | Mergesort | Quicksort |
| Worst case | O(N^2) | O(N^2) | O(N log N) | O(N^2) |
| Average case | O(N^2) | O(N^2) | O(N log N) | O(N log N) |
| Best case | O(N^2) | O(N) | O(N log N) | O(N log N) |
| In-Place? | Yes | Yes | No | Yes |
| Stable? | No | Yes | Yes | No |
| Adaptive? | No | Yes | No | No |
|  |  |  |  |  |

Selection sort is O(N^2)

(N^2 – N) / 2 comparisons (calls to less)

N – 1 exchanges (calls to swap)

Selection sort is not adaptive to its input, so all arrangements of data in the array will require a quadrative amount of work.

O(N^2) for already sorted, almost sorted, reverse, random

Insertion sort is O(N^2)

<= (N^2 – N) / 2 comparisons (calls to less)

<= (N^2 – N) / 2 exchanges (calls to swap)

Insertion sort is adaptive, so input that is sorted or almost sorted will only require a linear amount of time.

O(N) already sorted

O(N) “almost” sorted

O(N^2) in reverse order

O(N^2) in random order

MergeSort is O(NlogN)

Divide and conquer

QuickSort is O(NlogN)

Divide and Conquer

Asymptotically optimal for comparison sorting, worst case is O(N^2)

Bag – no particular order, duplicates allowed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Bag | Bag | Set | Set | Set | Set |
| method | ArrayBag | LinkedBag | Array | Nodes | Ordered Array | Ordered Nodes |
| Add | O(1)\* | O(1) | O(N) | O(N) | O(N) | O(logN)1 |
| Remove | O(N) | O(N) | O(N) | O(N) | O(N) | O(logN)1 |
| Contains | O(N) | O(N) | O(N) | O(N) | O(logN) | O(logN)1 |
| Size() | O(1) | O(1) | O(1) | O(1) | O(1) | O(1) |
| isEmpty() | O(1) | O(1) | O(1) | O(1) | O(1) | O(1) |
| Iterator | O(1) | O(1) | O(1) | O(1) | O(1) | O(1) |

\*amortized

1 = if using binary search O (logN), else O(N)

A list is a collection that keeps its elements in some particular order.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | IndexedList | IndexedList | ItemizedList | ItemizedList | SortedList | SortedList |
| Method | Array | Nodes | Array | Nodes | Array | Nodes |
| Remove | O(N) | O(N) | O(N) | O(N) | O(N) | O(logN) |
| addAfter | \* | \* | O(N) | O(N) | \* | \* |
| Add(element) | O(1) | O(1) | \* | \* | O(N) | O(logN) |
| Add(index, element) | O(N) | O(N) | \* | \* | \* | \* |
| Get | O(1) | O(N) | \* | \* | \* | \* |
| indexOf | O(N) | O(N) | \* | \* | \* | \* |

If we could use binary search on a node based structure, we could add, remove, and search a sorted list in O(log N) time.

A queue is a collection that maintains its elements in first-in, first-out (FIFO) order.

A stack is a collection that maintains its elements in last-in, first-out order (LIFO).

Pen and paper strategy for a fully parenthesized infix:

Scan from left to right:

When you get to a ) token:

Evaluate the most recent subexpression

Store the result for later

Pen and paper strategy for postfix:

Scan from left to right:

When you get to an operator token:

Evaluate using the most recent values

Store the result for later

Pen and paper strategy for prefix:

Scan from RIGHT to LEFT

When you get to an operator token:

Evaluate using the most recent values

Store the result for later

A tree is a collection in which elements are arranged in a hierarchy.

Height = length of the longest path from a given node to descendent leaf.

Full tree – all leaves have the same depth and every parent node has the maximum number of children.

Complete tree – A tree is complete if it is full to the next-to-last level, and the leaves on the lowest level are “left justified”

A full or complete tree is the shortest possible tree (minimum height) that could store N nodes.

Balanced tree – a tree is balanced if for each node, it’s subtrees have similar heights.

A balanced tree will have near-optimal height for storing N nodes.

Binary tree traversal:

Preorder: NLR

Postorder: LRN

Inorder: LNR

AVL Tree Balance Factor:

Bf = Height(Right) – (Height(Left)

AVL trees offer guaranteed O(log N) across all major operations

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Array | Linked List | AVL Tree |
| add | O(N) | O(N) | O(log N) |
| remove | O(N) | O(N) | O(log N) |
| Search | O(log N) | O(N) | O(log N) |