Jorge Solis
jas2430
05/01/2017
Professor Paul Blaer

Problem Set 5 – Written Component

7.19 – quicksort([3 , 1, 4 , 1 , 5 , 9 , 2 , 6 , 8 , 3 , 5])

3	1	4	1	5	9	2	6	5	3	5
3	1	4	1	5	9	2	6	5	3	5
3	1	4	1	5	5	2	6	5	3	9
3	1	4	1	5	3	2	6	5	5	9
3	1	4	1	5	3	2	5	5	6	9
3	1	4	1	5	3	2	5	5	6	9
1	1	4	3	5	3	2	5	5	6	9
1	1	4	2	5	3	3	5	5	6	9
1	1	4	3	5	2	3	5	5	6	9
1	1	2	3	5	4	3	5	5	6	9
1	1	2	3	5	4	3	5	5	6	9
1	1	2	3	5	4	3	5	5	6	9
1	1	2	3	3	4	5	5	5	6	9
1	1	2	3	4	3	5	5	5	6	9
1	1	2	3	3	4	5	5	5	6	9
1	1	2	3	3	4	5	5	5	6	9
1	1	2	3	3	4	5	5	5	6	9

7.23

Selecting the middle element of an array as a pivot functions identically to selecting the first element of an array if the array is completely randomly sorted. If the array is partially sorted or in reverse order, however, selecting the middle element of an array for use as a pivot is much better than selecting the first. Selecting the middle element of a partially-sorted or reverse-ordered array is much more likely to have equal amounts of larger and smaller elements, resulting in a runtime closer to the lower bound of NlogN.

9.1

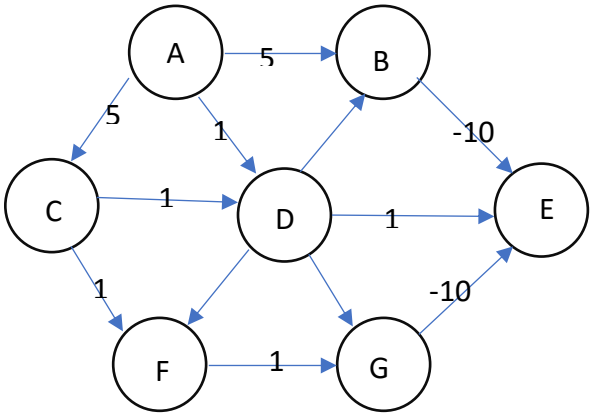
s,G,D,A,B,H,E,I,F,C,t

9.7a –

A->E Dijkstra: A, D, E = 2

Correct: A, B, E = -5

	A	B	C	D	E	F	G
A	-	5	5	1	∞	∞	∞
B	∞	-	∞	∞	-10	∞	∞
C	∞	∞	-	1	∞	1	∞
D	∞	5	∞	-	1	5	5
E	∞	∞	∞	∞	-	∞	∞
F	∞	∞	∞	∞	∞	-	1
G	∞	∞	∞	∞	-10	∞	-



7.28a – This method uses the median3() and swapReferences() methods from the Weiss textbook.

```
public void quicksort(AnyType[] a, int left, int right,)
{
	if(left + 3 >= right)
	{
		AnyType pivot = median3(a, left, right);
		int i = left, j = right – 1; leftDuplicates = 0; rightDuplicates = 0;

		for( ; ; )
		{
			while(a[++i].compareTo(pivot) < 0){}
			while(a[--j].compareTo(pivot) > 0){}
			if(i < j)
			{
				swapReferences(a, i, j);
				if(a[i].compareTo(pivot) == 0)
				{
					swapReferences(a,left++,i);
					leftDuplicates += 1;
				}
				if(a[j].compareTo(pivot)==0)
				{
					swapReferences(a,--right-1,j);
					rightDuplicates += 1;
				}
			}
			else
			{
				if(a[i].compareTo(pivot) == 0)
				{
					swapReferences(a, left++, j);
					leftDuplicates += 1;
				}
				if(a[j].compareTo(pivot) == 0)
				{
					swapReferences(a, --right - 1, i);
					rightDuplicates += 1;
				}
				break;
			}
		}
		left -= leftDuplicates;
		right += rightDuplicates;
		for(int k = 0; k < leftDuplicates; k++)
			swapReferences(a, left + k, j - k);
		for(int k = 0; k < rightDuplicates + 1; k++)
			swapReferences(a, i + k, right – 1 - k);
		quicksort(a, left ,j - leftDuplicates);
		quicksort(a, i + rightDuplicates + 1, right);
	}
	else
		insertionsort(a);
}
```

9.38a

```
public void compare(Stick B) {
        int c = ((b.x2 – b.x1)*(a.y1 – b.y1) – (b.y2 – b.y1)*(a.x1 – b.x1)) /
                ((b.y2 – b.y1)*(a.x2 – a.x1) – (b.x2 – b.x1)*(a.y2 – a.y1));
        int d = ((a.x2 – a.x1)*(a.y1 – b.y1) – (a.y2 – a.y1)*(a.x1 – b.x1)) /
                ((b.y2 – b.y1)*(a.x2 – a.x1) – (b.x2 – b.x1)*(a.y2 – a.y1));
        if( a.x2 – a.x1 > c*(a.x2 – a.x1)) {
                int aInt = a.z1 + c*(a.z2 – a.z1);
                int bInt = b.z1 + d*(b.z2 – b.z1);
                if (bInt > aInt)
                        b.sticksBelow(a);
                else
                        a.sticksBelow(a)
        }
        pass;
}
```

This algorithm checks if stick A is below stick B. It assumes that each stick object has a list of sticks below it, and an instance value sticksAbove equal to the number of sticks above it. There is a method sticksBelow() that adds the explicit argument to the list of the implicit argument, and increments the sticksAbove variable of the explicit argument.

- It first determines the intersection of stick A and stick B on the X,Y plane and then in X,Y,Z space. If stick B and stick A do not intersect in the X,Y plane, this method ends.

- If stick B and stick A do intersect in the X,Y plane, the method compares the z-coordinates of the respective sticks at the X,Y point of the intersection.
- If the z-coordinate of stick B at the intersection is greater than that of stick A, stick A is below stick B, so stick A is added to a list of sticks below stick B within the stick B data object.
- If the z-coordinate of stick A at the intersection is greater than that of stick B, stick B is added to a list of sticks below stick A within the stick A data object.

9.38b

This method can be used on a pile of sticks to return the order in which the sticks can be removed.

- The algorithm is run on every stick for every stick, with each stick object containing a list of sticks that are above it, which should require ON^2 time to carry out.
- After this, however, determining the order in which the sticks can be removed will take ONlogN time. Iterate through the list of sticks to find sticks with 0 sticks above them, and add them to a queue.
- Iterate across the list of stick objects below each stick with no sticks above it, and decrement the number of sticks above each by 1.
- Whenever a stick object's sticksAbove variable is decremented to 0, add it to the queue.
- If the queue has emptied but there are still stick objects on the ground, there is a cycle. If there are no cycles, there will be no sticks left on the ground when the queue empties.