HW5 – Written

7.19

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 | Move pivot to right end |
| 3 | 1 | 4 | 1 | **5** | 9 | 2 | 6 | 5 | **3** | 5 | Swap |
| 3 | 1 | 4 | 1 | 3 | **9** | 2 | 6 | **5** | 5 | 5 | Swap |
| 3 | 1 | 4 | 1 | 3 | 5 | **2** | **6** | 9 | 5 | 5 | Intersection. Swap pivot |
| 3 | 1 | 4 | 1 | 3 | **5** | **2** | 5 | 9 | 5 | 6 | Recurse left |
| 3 | 1 | 4 | 1 | 3 | 5 | 2 |  |  |  |  | Move pivot to right end |
| **3** | **1** | 4 | 2 | 3 | 5 | 1 |  |  |  |  | Swap |
| **1** | **3** | 4 | 2 | 3 | 5 | 1 |  |  |  |  | Intersection. Swap pivot |
| 1 | 1 | 4 | 2 | 3 | 5 | 3 |  |  |  |  | Insertion sort the left side |
| 1 |  |  |  |  |  |  |  |  |  |  | Recurse right |
|  |  | 4 | 2 | 3 | 5 | 3 |  |  |  |  | Move pivot to right end |
|  |  | **4** | 2 | **3** | 5 | 3 |  |  |  |  | Swap |
|  |  | 3 | **2** | **4** | 5 | 3 |  |  |  |  | Intersection. Swap pivot |
|  |  | 3 | 2 | 3 | 5 | 4 |  |  |  |  | Insertion sort the left side |
|  |  | 2 | 3 |  |  |  |  |  |  |  | Insertion sort the right side |
|  |  |  |  |  | 4 | 5 |  |  |  |  | Insertion sort the right side |
|  |  |  |  |  |  |  |  | 6 | 9 | 5 | Swap |
|  |  |  |  |  |  |  |  | 5 | 6 | 9 | DONE |
| 1 | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 9 |  |

7.23

Yes. If the array is perfectly sorted, then taking the first element as pivot would result in quadratic time. If the array is perfectly reverse-sorted, then taking the last element as pivot would result in quadratic time. For arrays partially sorted, which is close to a random arranged array, the middle position is likely to be close to the optimal pivot (the actual median), giving an average runtime of O(n) = n\*log(n).

Quadratic time is thus unlikely, but yet possible.

7.28a

void quicksort(a, l, r){

i = l

j = r - 1

el = i

er = j

while true do

while a[i] <= a[r] and i != r do

if a[i] == a[r] then

swap a[el] with a[i]

el = el + 1

i = i + 1

while a[j] >= a[r] and j != l do

if a[i] == a[r] then

swap a[er] with a[j]

er = er - 1

j = j - 1

if i < j then

swap a[i] with a[j]

else

break

swap a[i] with a[r - 1]

k = l

for j = el to i - 1

swap a[j] with a[k]

k = k + 1

k = r

for j = er to i + 1

swap a[j] with a[k]

k = k - 1

quicksort(a, l, i - 1)

quicksort(a, i + 1, r)

}

9.1

s G D A H B E I F C t

9.7a

Let there be an undirected edge such that it goes from a to b with weight w. If there are 3 edges:

[b, a, 2]

[b, c, 3]

[c, a, -2]

We use Dijstra’s algorithm from source b. Vertex b can see a and c. It defines a.distance = 2; a.prev = b; c.distance = 3; c.prev = b. Then it chooses the smallest one, a. From a, the smallest is c. It defines c.distance = 0 and c.prev = a since that is the shorter path.

This means that the shortest path produced will be b-a-c with a distance of 0 since 2 – 2 = 0. This is the wrong shortest path.

9.38

a.

Let the coordinates of stick a be (e, f, g), (i, j, k) and b be (p, q, r), (s, t, u). Now, we first check if they intersect in the xy plane. For that plug in z = 0. Then derive equations for a and b in the xy plane:

f = me + c

j = mi + c

and

q = np + k

t = ns + k

Then, we solve this system of equations and find the solutions for sticks a and b. If a solution exists, then the stick with the higher z coordinate: max(g,k) vs max(r,u). The stick with the higher z coordinate is the stick on top.

If no solution exists, it means that the sticks are unrelated.

b.

Let every Stick have the same interface as a vertex in a graph on N vertices.

for every Stick a in sticks

for every Stick b in sticks

if above(a,b) == a then

a.adjacentEdges.add(new Edge(a,b))

else if above(a,b) == b then

b.adjacentEdges.add(new Edge(b,a))

else

nothing

if cycleExists(sticks) then

return null //cannot pick all the sticks

else

return topologicalSort(sticks)

For the topologicalSort method we can use the method described in part a.

to check if cycle exists, we start from every vertex and try all the possible edges

recursively. If we reach the first vertex then return true, else return false.