Tyler Richard

CS 233 01

HW 03

1.
Numbers below buckets representing chaining

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	3		0	12			9	70		

1 42
98

Hashkey(1) = 4 collision, chaining
Hashkey(42) = 7 collision, chaining
Hashkey(98) = 4 collision, chaining

Linear

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	3		0	12	1	98	9	42	70	

Hashkey(1) = 4, collision, incremented until index 5
Hashkey(42)= 7 , collision, incremented until index 8
Hashkey(98)= 4 , collision, incremented until index 6
Hashkey(70)= 8 , collision, incremented until index 9

Quadratic

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	42		0	12	3		9	70	1	98

2.
500
Because the number of keys is not given therefore the largest number of buckets would be most beneficial to attempt to maintain a 0.75 load factor

3.

53491/106963 = 0.500
If using linear probing it is not necessary as the load factor is under .75, but yes you could rehash for better time performance.
If using separate chaining, no, as the load factor is < 0.75, rehashing is not necessary given that there would be no time benefit

4.

Insert()	O(n)
Rehash()	O(n)
Remove()	O(n)
Contains()	O(n)

Worst cases

7. The problem is the copying of the array - at twice the size of the oldArray. This gets expensive during rehashing when you fill over half of the cells.

8.

Push()	O(1)
Top()	O(1)
Pop()	Log(n)
BuildHeap()	O(n)

9.
A good example of an application for priority queues are operating systems. They would be used for load balancing. By prioritizing tasks by length of time required so that long, potentially unnecessary tasks do not obstruct short tasks of varying importance. Further, they can be used in interrupt handling. They can do so by prioritizing an interrupting task relative to an ongoing process.

11.

# Inserted									
10	10								
12	10	12							
1	1	12	10						
14	1	12	10	14					
6	1	6	10	14	12				
5	1	6	5	14	12	10			
15	1	6	5	14	12	10	15		
3	1	3	5	6	12	10	15	14	
11	1	3	5	6	12	10	15	14	11

12.

15	14	10	12	6	5	1	3	11

13.

15	14	11	12	6	5	10	3	
15	14	11	12	6	5	10		
15	14	11	12	6	10			

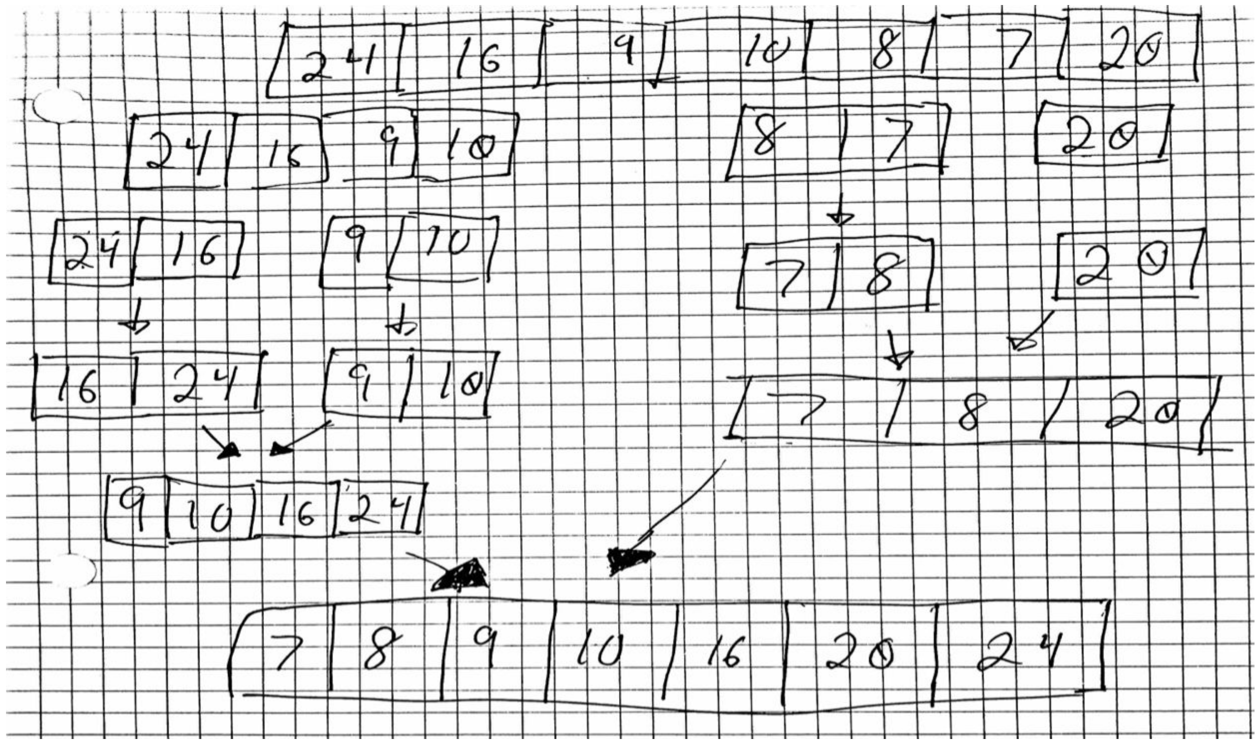
14.

Algorithm	Avg Complexity	Stable (Y/N)
Bubble Sort	O(n^2)	Y
Insertion Sort	O(n^2)	Y
Heap sort	O(Nlog(n))	N

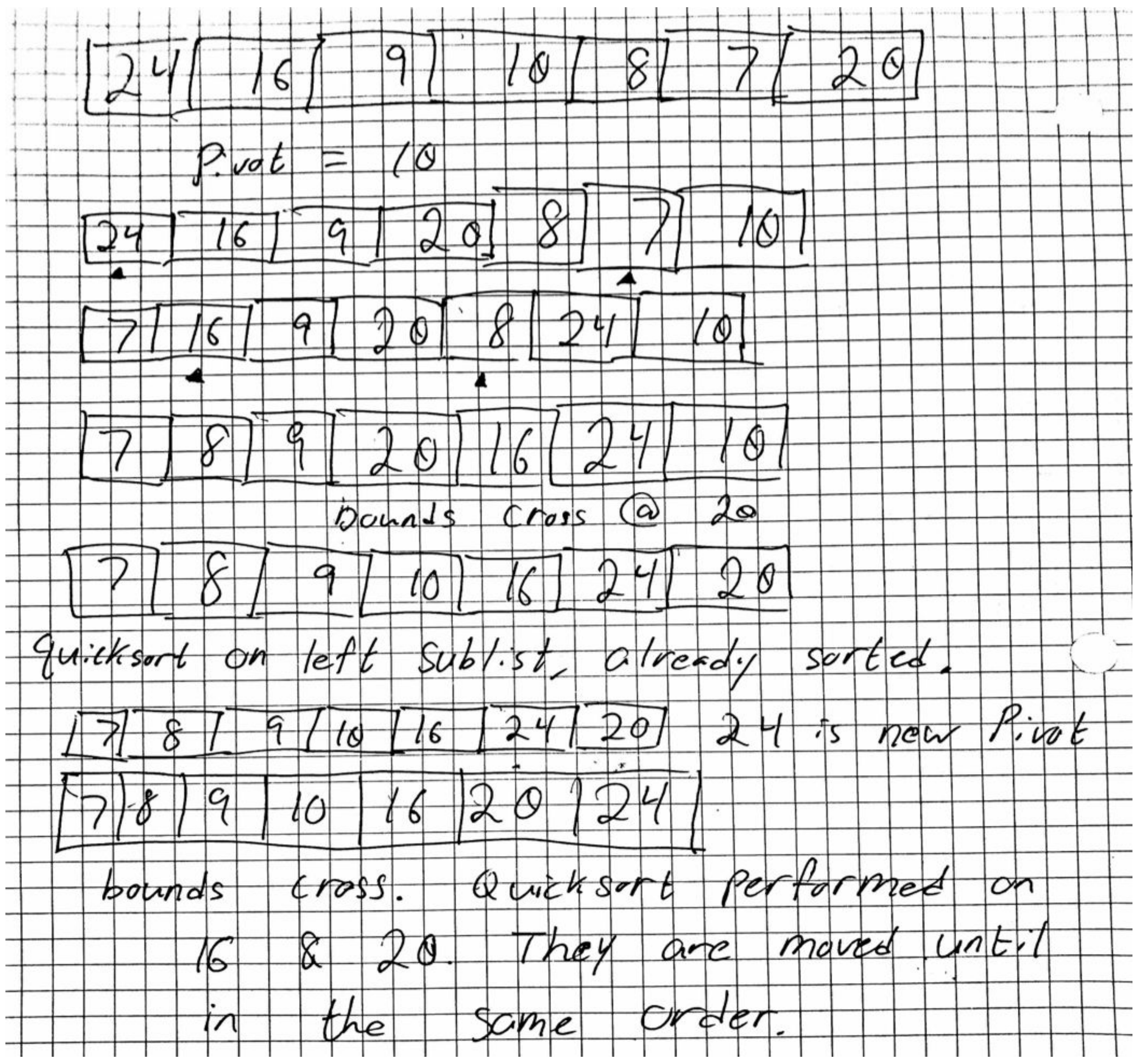
Merge sort	O(Nlog(n))	Y
Radix sort	O(n)	Y
Quicksort	O(Nlog(n))	N

15.
Quicksort compares values to a pivot, requires no extra memory, and does not preserve the relative order of elements. Mergesort divides the array into two subarrays repeatedly until one array is left, and requires extra memory. Languages choose one or the other primarily with concern to whether or not the algorithm is stable and memory requirement

16.



17.



24	16	9	10	8	7	20

Pivot = 10

24	16	9	20	8	7	10

7	16	9	20	8	24	10

7	8	9	20	16	24	10

bounds cross @ 20

7	8	9	10	16	24	20

quicksort on left sublist, already sorted.

7	8	9	10	16	24	20

24 is new Pivot

7	8	9	10	16	20	24

bounds cross. Quicksort performed on 16 & 20. They are moved until in the same order.

18
Quicksort