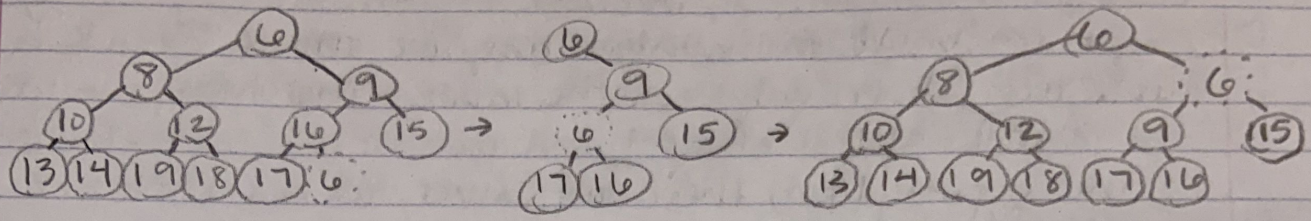
HW 3

Lauren Lyons

1 ▷ A = [6, 8, 6, 10, 12, 9, 15, 13, 14, 19, 18, 17, 16]



2 ▷ An algorithm that has O(nlgn) and converts A into a set can be built off of the mergesort algorithm. When mergesort stops partioning the array and starts to compare and place the numbers, we can add another comparison to check for duplicates. If there is a duplicate, we would then remove it and shift the iterations.

3 ▷ Heapsort is an O(nlgn) algorithm that can be used as a base for an algorithm that finds the mode of a sequence of numbers. The extra data we will use is a tuple that will hold the current mode and the number of times it occured. We will initialize that to the first element and give it a 1 for occurrence times. We will also have a local count variable that counts the current elements occurrence rates. So, as heapsort pops elements off of the heap, this is when we will count because the numbers that are the same will pop off right after one another, giving us the chance to individually count them. Then when the reoccurring element changes, we compare the current mode and its occurrence times with the current element and its local count variable. If it occurs more, then the mode is replaced. In the end, when the heapsort is done, the resulting tuple should hold the mode.

Lauren
Lyons

4 A stable algorithm will keep the equal elements in the same order in the output. So, in order for this to apply to any comparison-based sorting algorithm, we should create a tuple for each value of the input. The tuple should contain the original element and then its original place/order in the input. Using these two values, we can first sort based off element values, and then for any elements that are the same, you can use their original order to place them in the output.

5 a) Quickselect(A, p, r, k) → Final return: 4

input A = [10, 3, 9, 4, 8, 5, 7, 6] A after partition = [3, 4, 5, 6, 8, 9, 7, 10]

p = 1, r = 8, k = 2 q = 4 pivotDistance = 4

↓ Quickselect(A, p, q-1, k) ↑ return 4

input A = [3, 4, 5, 6, 8, 9, 7, 10] A after partition = [3, 4, 5, 6, 8, 9, 7, 10]

p = 1, r = 3, k = 2 q = 3 pivotDistance = 3

↓ Quickselect(A, p, q-1, k) ↑ return 4

input A = [3, 4, 5, 6, 8, 9, 7, 10] A after partition = [3, 4, 5, 6, 8, 9, 7, 10]

p = 1, r = 2, k = 2 q = 2 pivotDistance = 2

↳ if k = pivotDistance then return A[q] = A[2] = 4

b) - The first base case checks whether the input array A only has one element or not. If their is one element, that one element will fulfill every k-th smallest number, so it will be returned.
- Only step 1 is executed. Both the comparison and the return statement execute.
- T(first base case) = 7 = T(1)

b/c read p, read r, compare, read p, find mem. loc, read value, return

Lauren Lyons

5cont) b cont) - The second base case is if the pivotDistance is the same as the k^{th} value, meaning we know the k^{th} value. The pivotDistance is returned from the partition call. Then we return A[q], which is that k^{th} smallest value.

- step 1 (only the comparison), step 2, 3, 4, and 5 are executed.
- T(second base case) = 20n + 15

b/c read p, read r, compare, read q, read p, subtract, add 1, assign, read k, read pD, compare, read q, find mem loc., read value, assign

- steps 1 (only the comparison), 2, 3, 4, 6, 7 or steps 1 (only the comparison), 2, 3, 4, 6, 8 will execute.
- T(n → worst case) = 20n + T(n-1) + 14

b/c read p, read r, compare, read q, read p, subtract, add 1, assign, read k, read pD, compare, read k, read pD, compare

-The worst case will be n-1 because the Partition call may pick the largest number as the pivot number, meaning q will be returned as n and then QuickSelect is called as QS(A, p, q-1, k) which makes the ending index n-1, meaning the input size is n-1 (assuming that p is 1).

- T(n → best case) = $20n + T(\frac{n-1}{2}) + 14$

The best case input size would be $\frac{n-1}{2}$. The q+1 and q-1 in the recursion calls make it n-1, then that divided by 2 is the best case because overall, that will give us the least amount of recursion calls to reach our final answer.

6) A: [19, 6, 10, 7, 16, 17, 13, 14, 12, 9]
(1 2 3 4 5 6 7 8 9 10)

B: [6, 7, 9, 10, 12, 13, 14, 16, 17, 19]
(1 2 3 4 5 6 7 8 9 10)

initial C: [0 0 0 0 0 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1]

added C: [0 0 0 0 0 0 1 2 2 3 4 4 5 6 7 7 8 9 9 10]

Final C: [0 0 0 0 0 0 0 1 2 2 3 4 4 5 6 7 7 8 9 9]
(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)

Lauren
Lyons

7 ▷

initial				Final
[4567	[3210	[3210	[3210	[2345
3210	4321	4321	4321	3210
2345	2345	2345	2345	4321
4321	4567	4567	4567	4567
5678]	5678]	5678]	5678]	5678]

8 ▷ length(A) = 15

B0	$[0, 0.0\overline{6})$
B1	$[0.0\overline{6}, 0.1\overline{3})$
B2	$[0.1\overline{3}, 0.2)$
B3	$[0.2, 0.2\overline{6})$
B4	$[0.2\overline{6}, 0.\overline{3})$
B5	$[0.\overline{3}, 0.4)$
B6	$[0.4, 0.4\overline{6})$
B7	$[0.4\overline{6}, 0.5\overline{3})$
B8	$[0.5\overline{3}, 0.6)$
B9	$[0.6, 0.6\overline{6})$
B10	$[0.6\overline{6}, 0.7\overline{3})$
B11	$[0.7\overline{3}, 0.8)$
B12	$[0.8, 0.8\overline{6})$
B13	$[0.8\overline{6}, 0.9\overline{3})$
B14	$[0.9\overline{3}, 1)$

B0	$[0, 1/n)$
B1	$[1/n, 2/n)$
B(n-2)	$[n-2/n, n-1/n)$
B(n-1)	$[n-1/n, n/n)$

9 ▷ Bucketsort is not an inplace algorithm because it requires extra memory space. It requires a B array to hold its linked lists.

Lauren
Lyons

10 To sort through these $[0, n^2-1]$ values, we must change them to numbers of base n rather than base 10. After doing that, we will place them in tuples of original value and base n values. We can sort by the base n values and then output the original values. We will use Radix sort to sort them, giving us $O(n)$ time.

Lauren
Lyons

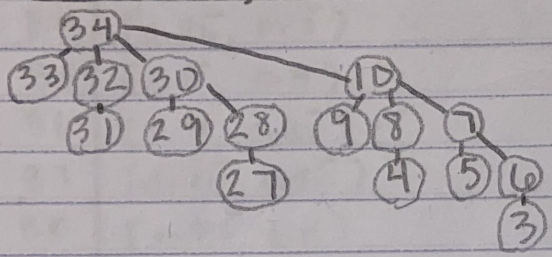
11

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Arbitrary	0	1	14	3	3	3	1	1	8	3	3	3	3	1	14	14	16	19	20	1
Height	-4	1	14	3	3	3	1	1	8	3	3	3	3	1	14	1	16	16	16	16
Size	3	1	-20	3	3	3	1	1	8	3	3	3	3	3	14	3	16	16	16	16
Path Comp	0	1	14	3	3	3	1	1	8	3	3	3	3	1	14	1	1	1	1	1

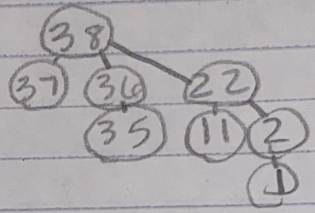
12

$\{B_3\ B_2\ B_1\}$ 1110
$+\{B_3\ B_1\}$ +1010
$\{B_4, B_3\}$ 11000

After Merge

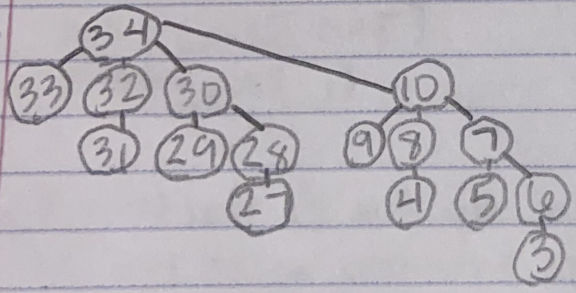


B_4

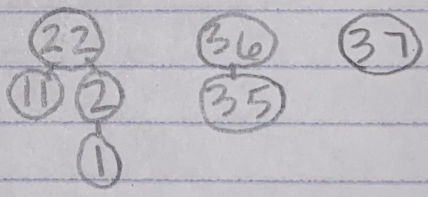


B_3

After Extract Max = 38



B_4



B_2 B_1 B_0