Hongdao Meng hm3424 Homework 2.

Q1 a. $h_1(k) = k^2 \bmod 13$ $h_2(k) = k^2+1 \bmod 13$ $pos = (h_1(k) + h_2(k) \cdot i) \bmod 13$

Input: 45, 10, 100, 17, 4, 26, 24, 6, 9

45: $h_1 = 45^2 \% 13 = 10$ $T[10] = 45$

10: $h_1 = 10^2 \% 13 = 9$ $T[9] = 10$

100: $h_1 = 100^3 \% 13 = 3$ $T[3] = 100$ $3+4=7$

17: $h_1 = 17^2 \% 13 = 3$ (conflision) $h_2 = (17^2+1) \% 13 = 4$ $T[7] = 17$

4: $h_1 = 4^2 \% 13 = 3$ (collision) $h_2 = (4^2+1) \% 13 = 4$ $3 \to 4 \to 7 \to 11$

$T[11] = 4$

26: $h_1 = 26^{20} \% 13 = 0$ $T[0] = 26$

24: $h_1 = 24^2 \% 13 = 4$ $T[4] = 24$ $(11+10) \bmod 13 = 8$

6: $h_1 = 6^2 \% 13 = 10$ (collision) $h_2 = (6^2+1) \% 13 = 11$ $T[8] = 6$

9: $h_1 = 9^2 \% 13 = 3$ (conllision) $h_2 = (9^2+1) \% 13 = 4$ $3 \to 7 \to 11 \to 2$ $(15 \% 13)$

$T[2] = 9$

∴ $T[0]=26$, $T[1]=6$, $T[4]=17$, $T[6]=45$, $T[7]=4$, $T[9]=100$, $T[10]=10$, $T[11]=24$ $T[12]=9$.

Yes, In double hashing, the final hash table may have a different structure with different inserted order.

This bcs double hashing uses fixed probe sequences for each key, but insertion order determines which slot are occpuied first. If a key's probe sequence overlaps with anthor key's path, eariler insertions block slots and force later keys to settle in different position

And All keys are inserted.

Q1b. $h(k,i) = k + 2i + 3i^2 \bmod 13$

45: $h(45,0) = 45\%13 = 6 \rightarrow T[6] = 45$

10: $h(10,0) = 10\%13 = 10 \rightarrow T[10] = 10$

100: $h(100,0) = 100\%13 = 9 \rightarrow T[9] = 100$

17: $h(17,0) = 17\%13 = 4 \rightarrow T[4] = 17$

4: $h(4,0) = 4$ (collision) $\rightarrow h(4,1) = 9 \rightarrow h(4,2) = 20\%13 = 7$

$T[7] = 4$

26: $h(26,0) = 0 \rightarrow T[0] = 26$

24: $h(24,0) = 11 \rightarrow T[11] = 24$

6: $h(6,0) = 6$ (conllision) $\rightarrow h(6,1) = 11 \rightarrow h(6,2) = 22\%13 = 9$

$\rightarrow h(6,3) = 39\%13 = 0 \rightarrow h(6,4) = 326\%13 = 1 \rightarrow T[1] = 6$

9: $h(9,0) = 9$ (conllision) $\rightarrow h(9,1) \rightarrow h(9,2) = 12 \rightarrow T[12] = 9$

∴ $T[0] = 26$, $T[1] = 6$, $T[4] = 17$, $T[6] = 45$, $T[7] = 4$, $T[9] = 100$, $T[10] = 10$, $T[11] = 24$, $T[12] = 9$.

All keys are inserted.

Q1c. $h(k) = k^2 \bmod 13$

45: $h(45) = 45^2\%13 = 10 \rightarrow T[10] = 45$

10: $h(10) = 10^2\%13 = 9 \rightarrow T[9] = 10$

100: $h(100) = 100^2\%13 = 9 \rightarrow T[9]: 10 \rightarrow 100$

17: $h(17) = 17^2\%13 = 3 \rightarrow T[3] = 17$

4: $h(4) = 4^2\%13 = 3 \rightarrow T[3]: 17 \rightarrow 4$

26: $h(26) = 26^2\%13 = 0 \rightarrow T[0] = 26$

24: $h(24) = 24^2\%13 = 4 \rightarrow T[4] = 24$

6: $h(6) = 6^2\%13 = 10 \rightarrow T[10]: 45 \rightarrow 6$

9: $h(9) = 9^2\%13 = 3 \rightarrow T[3]: 17 \rightarrow 4 \rightarrow 9$

∴ $T[0] = 26$

$T[3]: 17 \rightarrow 4 \rightarrow 9$

$T[4] = 24$

$T[9] = 10 \rightarrow 100$

$T[10] = 45 \rightarrow 6$

Q1d. key 6.

double hashing: $h_1 = 10 \rightarrow$ [redacted] (collision) $h_2 = 11 \rightarrow T[8] = 6$ 2 probes

quadratic hashing: $i = 0 \rightarrow 6$, $i = 1 \rightarrow 11$, $\cdots i = 10 \rightarrow T[1] = 6$, 11 probes

chaining hashing: $T[10] : 45 \rightarrow 6$ 2 probes

Q1e. $h(k,i) = k + 2i + 2i^2 \mod 7$

$T[0] = 7$, $T[3] = 3$, $T[4] = 4$, $T[5] = 5$

$h(k,i) = k + 2(i+i)^2 \mod 7$

need a key, $h(k,i) \rightarrow 0 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0$ or other loop.

if $k = 0$

$i = 0$	$h = 0$
$i = 1$	$h = 4$
$i = 2$	$h = 5$
$i = 3$	$h = 3$
$i = 4$	$h = 5$
$i = 5$	$h = 4$
$i = 6$	$h = 0$

a loop: $0 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 0$.

can never get 1, 2, 6, so key=0 fails to insert.

This is because the orders made by $2i + 2i^2$ can not cover all slots.

We can use double hashing to find a free spot in the table.

Q2 a. Original n=55 k=14

Step 1. Find Median of Medians (Groups of 5)

G1 [25, 37, 9, 52, 14] → [9, 14, 25, 37, 52] → Median = 25

G2 → [7, 19, 35, 42, 89] → Median = 35

G3 → [4, 15, 31, 53, 83] → Median = 31

G4 → [40, 46, 66, 86, 99] → Median = 66

G5 → [2, 5, 8, 22, 34] → M = 8

G6 → [10, 18, 30, 68, 90] → M = 30

G7 → [17, 20, 21, 29, 84] → M = 21

G8 → [12, 16, 33, 45, 77] → M = 33

G9 → [6, 19, 41, 42, 53] → M = 41

G10 → [3, 18, 23, 91, 93] → M = 23

G11 → [111, 86, 87, 88] → M = 86

3.1 Divide into groups of 5.

3.2 Sort each group, using insertion sort.

3.3 Find the median of the medians.

Medians [25, 35, 31, 66, 8, 30, 21, 33, 41, 23, 86] Pivot = ~~30~~ ~~28~~ 31

Step 2. Compare each element to Pivot = 31

Using Partition algorithm.

Element ≤ 31 (~~28~~ elements)

[25, 9, 14, 7, 19, 15, 4, ~~31~~, 5, 22, 2, 8, 10, 18, 30, 21, 17, 20, 29, 12, 16, 6, 19, 3, 23, 18, 1, 11]

Rand of Pivot (31) r = 28 ∵ k < r ∴ Recurrence on left subarray.

Step 3. Recursive Call on left subarray (n=28)

G1 → [7, 9, 14, 19, 25] → M = 14

G2 → [4, 5, 15, 22, 31] → M = 15

G3 → [2, 8, 10, 18, 30] → M = 10

G4 → [21, 17, 20, 29, 12] → [12, 17, 20, 21, 29] → M = 20

G5 → [3, 6, 16, 19, 23] → M = 16

G6 → [1, 11, 18] → 11

Pivot = 14

Median [10, 11, 14, 15, 16, 20] M = 14

Step 4. Partition around Pivot = 14

Element ≤ 14 (12 elements)

[9, 7, 4, 5, 2, 8, 10, 12, 6, 3, 1, 11]

Pivot r = 13 [14]

∵ k > r ∴ Recurrence on right subarray

Element > 14 [25, 19, 15, 22, 18, 30, 21, 17, 20, 29, 16, 19, 23, 18]

Step 5. Recursive Call on Righe subarray (n=14) k=14-13=1

G1 → [15, 18, 19, 22, 25] → M=19

G2 → [17, 20, 21, 29, 30] → M=21

G3 → [16, 18, 19, 23] → M=18

Median array [18, 19, 21]

Pivot = 19.

Step 6. Partition around pivot = 19 k=1

Elements ≤ 19 (5 elements)

[14, 15, 18, 17, 16, 19, 18] r=6 ∵ k < r ∴ on Left

Step 7. Recursive Call on Left subarray n=5 k=1

G1 → [14, 15, 17, 18, 18] → 16 k=1 : 15

~~G2 → [12, 16, 18, 19] → 16~~ ~~M=16 Pivot=16~~

∵ k=1 ∴ We find 15

~~Step 8. Partition around Pivot=16 k=1~~

~~Elements ≤ 16~~

~~[14, 15, 12, 16] r=4 k<r ∴ on left subarray.~~

~~Step 9. Recursive Call on left subarray~~

~~G1 → [12, 14, 15, 16] M=14=Pivot~~

∴ 15 is the element of rank 14.

Q2b.

Step by Step Runtime Analysis

1. sort each group of size 7. Group = $n/7$, each group's runtime is constant,

   $\therefore O(\frac{n}{7} \cdot c)$ is $O(n)$

2. Find median of the medians $O(\frac{n}{7} \cdot c)$ is $O(n)$ $T(\frac{n}{7})$ ← Recursive Call

3. Partition around pivot

   Time: $O(n)$

4. $k = r$ return x $O(1)$

5. Recurse : $T(\frac{11}{14}n)$ $\frac{n}{7} \cdot \frac{1}{2} \cdot 3 = \frac{3n}{14} < x$

   $n - \frac{3n}{14} = \frac{11}{14}n$

$\therefore T(n) \leq T(\frac{n}{7}) + T(\frac{11}{14}n) + O(n)$

Substitution Method: Goal: ~~$T(n) \leq T(\frac{n}{7}) + T(\frac{11}{14}n) + O(n)$~~ is $O(n)$

$T(n) \leq c \cdot n$

$\because T(\frac{n}{7}) \leq$ ~~$T(\frac{n}{49}) + T(\frac{11}{14 \times 7}n) + O(n)$~~ $\frac{c}{7} \cdot n \leq cn$

$T(\frac{11}{14}n) \leq \frac{11}{14} \cdot cn \leq cn$

$T(n) \leq (\frac{c}{7} + \frac{11}{14}c + a) \cdot n \leq (\frac{13}{14}c + a)n \leq c^* n$

* if $\frac{13}{14}c + a \leq c$ $a \leq \frac{c}{14}$

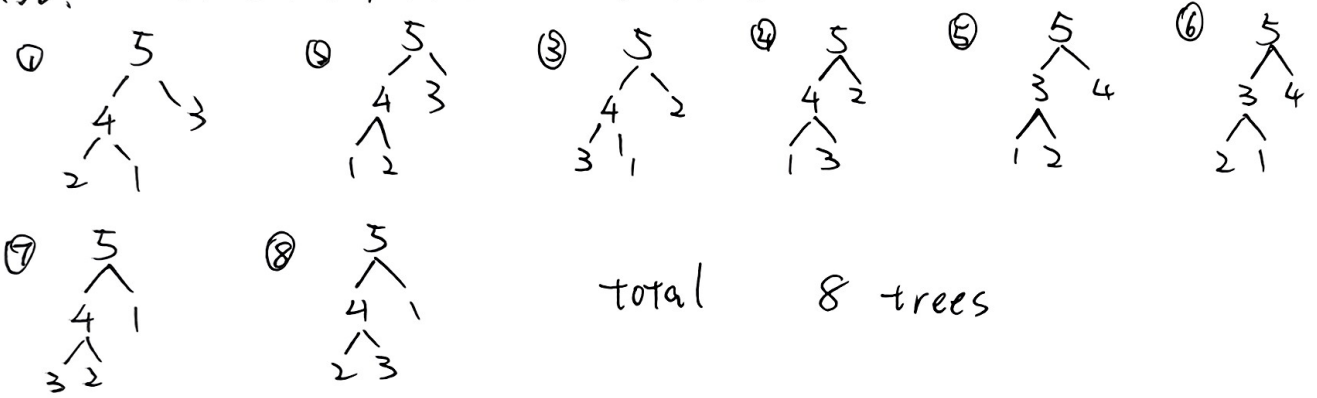
$c \geq 14a$

has $T(n) \leq cn$ $\therefore T(n)$ is $O(n)$

Q3. Heap

Q3a. No, this algo doesn't correctly build a max-heap. Because this algo precesses nodes from the ~~roo~~ root downward, which may leave subtrees invalid after parent adjustment, leading to an incorrect heap. And the right way is start from the last non-leaf node and moving upward to ensure the subtrees are valid heaps.

Q3b. 1, 2, 3, 4, 5 Root = 5



Q3c.

① many insert, a few remove_max

heap: insert is O(logn) remove is O(logn)
unordered array: insert O(1) remove O(n)
ordered array: insert O(n) remove O(1)

∴ Use an unordered array because inserts are O(1) and remove is O(n).

② many find_max, a few insert and remove_max

heap: find O(1) other O(logn)
unordered: find O(n) insert O(1) remove O(n)
ordered: find O(1) insere O(n) remove O(1)

Use a heap with find_max O(1). And ~~this~~ heap maintains O(logn) inseres/removes, which is acceptable for low frequency operations.

Q3 d

```
MinMaxInsert(A, k)
    A.heapsize ++
    A[A.heapsize] = k
    L =
    i = A.heapsize
    if Li is even:  # min level
        if A[parent] < A[i]:
            Swap A[parent] and A[i]
            MinBubbleUp(A, parent)
        else:
            MaxBubbleUp(A, i)
    else  # max level
        if A[parent] < A[i]:
            swap A[parent] and A[i]
            MaxBubbleUp(A, parent)
        else
            MinBubbleUp(A, i)
```

```
MinBubbleUp(A, i)
    if i != 1
        parent = i//2
        grand = parent//2
        if A[i] < A[grand]
            swap A[i] and A[grand]
            MinBubbleUp(A, grand)
```

```
MaxBubbleUp(A, i)
    if i != 1
        parent = i//2
        grand = parent//2
        if A[i] > A[grand]
            swap A[i] and A[grand]
            MaxBubbleUp(A, grand)
```

Q4a. radix sort (d, r)

A : n natural numbers, binary, at most 4^n

step 1. $d = \log_2(4^n) + 1 = 2n+1$ $\quad r = 2$

step 2. Using counting sort to sort by digit $\rightarrow O(n)$

From lowest to highest (total $2n+1$), counting sort bit by bit.

repeat step 2 by $(2n+1)$ times $\quad \therefore$ Runtime : $O(n \cdot (2n+1) = O(n^2)$

B : n integers, range $-n^2, \cdots, n^2$

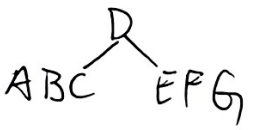
step 1 All elements plus $n^2 \rightarrow 0, \cdots, 2n^2$

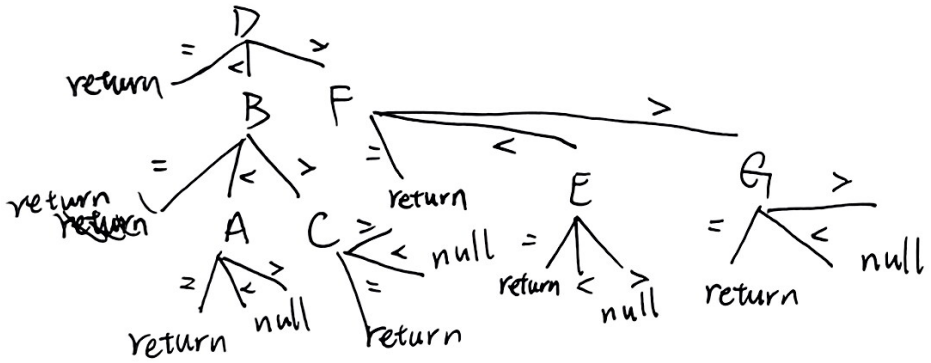
step 2. set $r = n$ $\quad 2n^2 = 2 \cdot n^2 + 0 \cdot n + 0 \quad d = 3$

step 3. From lowest to highest (total 3), counting sort digit by digit.

Repeat step 3, 3 times, each time : $O(n)$, and total runtime : $O(n)$

Q4b. [A, B, C, D, E, F, G] binary search key k





in each node compare with k.

Q4c. The exceptruntime is $O(n)$. The premise is that input data is evenly distributed. At that ~~situation~~, ~~eac~~ elements in each buckee roughly $O(1)$. However, if input isn't uniformly distributed, this ~~assumption~~ failed.

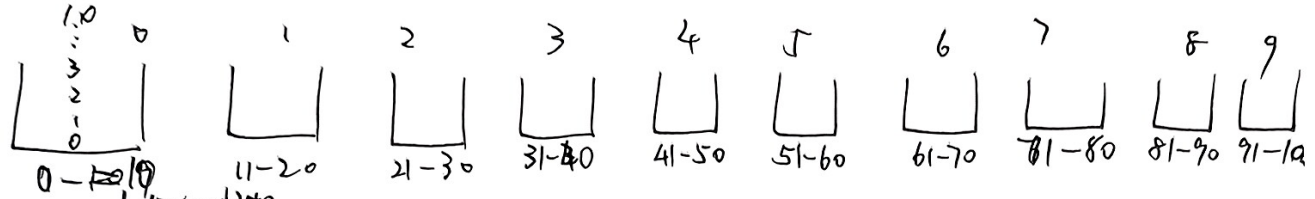
And Algorithm's performance degrades due to unbalanced buckee sizes, making worst-case runtime significantly slower than $O(n)$.

In worst case, $O(\Sigma n_i^2) + O(n) = O(n^2)$

$n_i = n$

Q4d. Input: 0 to 100.

Use 10 buckets



step1. Insert and disturbute the integer to the ~~bucket~~ corresponding bucket. like k to k/10 bucket $O(n)$

step2. Sort ~~each~~ lists in buckets $O(1)$

step3 Iterate through bucket 0 to 9, retrieving sorted elements in turn and merge into a sorted array. $O(n)$

Runtime: $O(n)$

∵ The Input is a set of uniform integers.

∴ Elements number in each bucket is $\frac{n}{10}$, and sorting time in bucket is linear

∴ Total runtime: $O(1 \cdot n) + O(n) = O(n)$