PROBLEM 1. The running times $T(n)$ of algorithms are defined by the following recurrent equations. Evaluate the running times using $\Theta$-notation.

(a) [5 POINTS] $T(n) = 4T(n/4) + 12$;

$a = 4 ; b = 4$

$f(n) = 12 \cdot \Theta(1) \Rightarrow 12 \cdot \Theta(n^0 \cdot \log^0(n))$

$k = 0 ; p = 0$

$\log_b a = \log_4 4 = 1$

$\log_4 4 = 1 > k = 0$

Case 1 : $\Theta(n^{\log_4 4}) = \Theta(n)$

---

(b) [5 POINTS] $T(n) = 25T(n/5) + n^2$;

$a = 25 ; b = 5$

$f(n) = \Theta(n^2) \Rightarrow \Theta(n^2 \cdot \log^0(n))$

$k = 2 ; p = 0$

$\log_b a = \log_5 25 = 2$

$\log_5 25 = 2 = k = 2$

Case 2 : $\Theta(n^2 \cdot \log(n))$

---

(c) [5 POINTS] $T(n) = 12T(n/3) + n^2$;

$a = 12 ; b = 3$

$f(n) = \Theta(n^2) = \Theta(n^2 \cdot \log^0(n))$

$k = 2 ; p = 0$

$\log_b a = \log_3 12 = \log_3(3 \cdot 4) \Rightarrow \log_3 3 + \log_3 4 \Rightarrow 1 + \log_3(2^2) = 1 + 2 \cdot \log_3 2$

$\log_3 12 = 1 + 2 \cdot \log_3 2 > k = 2$

Case 1: $\Theta(n^{\log_3 12}) = \Theta(n^{1 + 2\log_3 2}) = \Theta(n^{2.26})$

---

(d) [5 POINTS] $T(n) = 16T(n/8) + 2\log n + n/3$.

$a = 16 ; b = 8$

$\log_b a = \log_8 16 = \log_{2^3} 16 = \frac{1}{3} \log_2 16 = \frac{1}{3} \cdot 4 = \frac{4}{3}$

$k = 2 ; p = 1$

$\log_8 16 = \frac{4}{3} > k = 1$

Case 1: $\Theta(n^{\log_8 16}) = \Theta(n^{4/3}) = \Theta(n^{1.33})$

---

PROBLEM 2. [10 POINTS] Suppose there are three options for dividing a problem of size $n$ into subproblems:

- ✓ Algorithm $A$ solves the problem by recursively solving eight instances of size $n/2$, and then combining their solutions in time $\Theta(n^3)$.
- ✓ Algorithm $B$ solves the problem by recursively solving twenty instances of size $n/3$, and then combining their solutions in time $\Theta(n^2)$.
- ✓ Algorithm $C$ solves the problem by recursively solving two instances of size $2n$, and then combining their solutions in time $\Theta(n)$.

Which one is preferable, and why?

Algorithm C is preferable because it has a time complexity of $\Theta(n)$. $\Theta(n) > \Theta(n^2) > \Theta(n^3)$. The next best choice would be Algorithm B. After that then Algorithm A.

PROBLEM 3. As you know, when implementing a hash table, we normally have hash collisions where multiple elements have the same hash value. Thus, we should try to choose such a hash function that will not result in too many collisions. Suppose you have an empty hash table of length 13 with hash function

$$h(x) = x \bmod n.$$

(a) [5 POINTS] How many collisions will be in the hash table for the following dataset (explain your answer!):

14, 27, 40, 53, 66, 79, 92, 105, 118, 131, 144?

There are 10 collisions

| | |
|---|---|
| 14 mod 13 = 1 ✓ | 92 mod 13 = 1 ✗ |
| 27 mod 13 = 1 ✗ | 105 mod 13 = 1 ✗ |
| 40 mod 13 = 1 ✗ | 118 mod 13 = 1 ✗ |
| 53 mod 13 = 1 ✗ | 131 mod 13 = 1 ✗ |
| 66 mod 13 = 1 ✗ | 144 mod 13 = 1 ✗ |
| 79 mod 13 = 1 ✗ | |

(b) [5 POINTS] Can you come up with another $n$ for which the function $h(x) = x \bmod n$ produces no collisions for this dataset without increasing the size of the hash table?

If your answer is "yes", give the value of $n$ and explain why there are no collisions. If your answer is "no", explain why it is impossible to do.

If $n = 12$ there will be no collisions. If memory space is a concern then $n = 11$ is a better choice.

| | |
|---|---|
| 14 mod 12 = 2 ✓ | 92 mod 12 = 4 ✓ |
| 27 mod 12 = 3 ✓ | 105 mod 12 = 9 ✓ |
| 40 mod 12 = 4 ✓ | 118 mod 12 = 10 ✓ |
| 53 mod 12 = 5 ✓ | 131 mod 12 = 11 ✓ |
| 66 mod 12 = 6 ✓ | 144 mod 12 = 0 ✓ |
| 79 mod 12 = 7 ✓ | |

(c) [5 POINTS] Propose a function different from $h(x) = x \mod n$ which produces no collisions for our dataset.

new hash = $(idx \cdot x) \mod (n + x)$
this hash function works but a more space efficient one would be more suitable

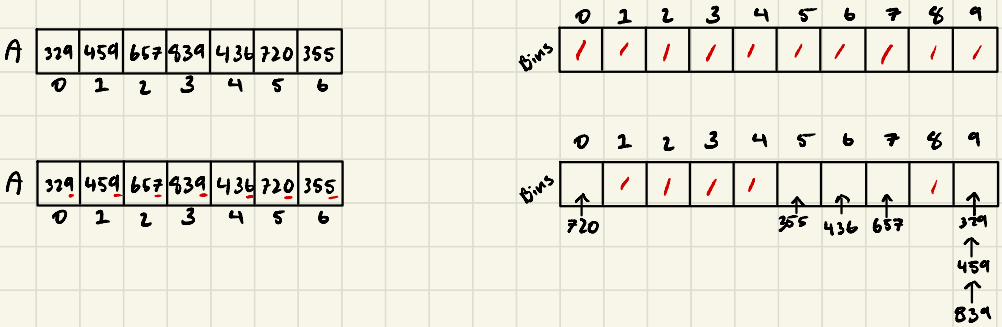
$(0 \cdot 14) \mod (11 + 14) = 0$ ✓
$(1 \cdot 27) \mod (11 + 27) = 27$ ✓
$(2 \cdot 40) \mod (11 + 40) = 29$ ✓
$(3 \cdot 53) \mod (11 + 53) = 31$ ✓
$(4 \cdot 66) \mod (11 + 66) = 33$ ✓
$(5 \cdot 79) \mod (11 + 79) = 35$ ✓
$(6 \cdot 92) \mod (11 + 92) = 37$ ✓
$(7 \cdot 105) \mod (11 + 105) = 39$ ✓
$(8 \cdot 118) \mod (11 + 118) = 41$ ✓
$(9 \cdot 131) \mod (11 + 131) = 43$ ✓
$(10 \cdot 144) \mod (11 + 144) = 45$ ✓

PROBLEM 4. [10 POINTS] Describe how the following integers would be sorted by a base-10 radix sort.

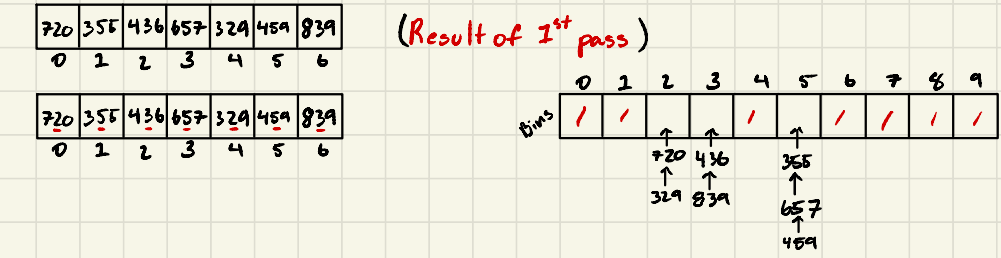
$(329, 457, 657, 839, 436, 720, 355) \longrightarrow (329, 355, 436, 457, 657, 720, 839)$

Using radix sort, the last digit of each element, in the set, will signify where each element goes in an array of size 10. The reason it is 10 is because that is the base. So 0 digit, as the last element will be referenced by Bin[0]. If it was a 9 as the last digit then that would be Bin[9]. If there is already a element at that index then enqueue it to the end of all the elements that are already there. Once you have done this for all the elements, pass through the arranged elements and put them in a array indexed the way they are recieved. Make sure to take out the elements in a FIFO style. Now do all these steps with the next digit over. You keep doing this until the element in the set, with the longest amount of digits has been reached. If a element doesn't have a digit where another element does the just put it in Bin[0].

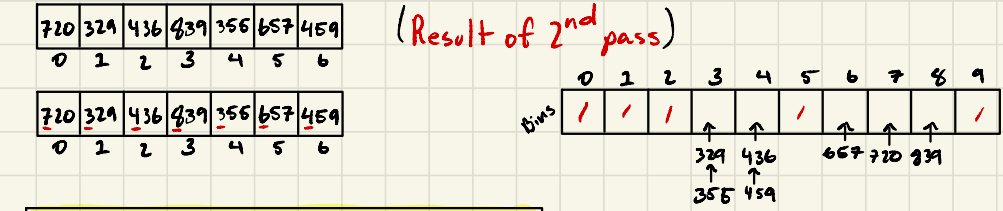
Example on Next Page

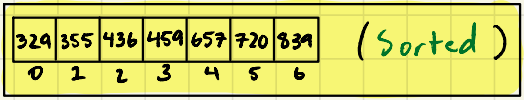



(Result of 1st pass)




(Result of 2nd pass)




(Sorted)

PROBLEM 5. [10 POINTS] Describe a linear time algorithm to sort a set $n$ of strings, each having $k$ English characters.

Radix Sort can sort this set of strings. Radix sort has a time complexity of $\Theta(d \cdot (n+k))$. The number of characters to represent each character equals $d$. The length of the set equals $n$. The length of each string equals $k$. $(n+k)$ is a counting sort for the position in the strings. It does this counting sort $d$ times.