

Total Marks: 45

- (1) (5 points) A “divide-and-conquer” algorithm is one which splits up the input into equal parts, recurses on these parts, and then puts the parts back together. Such an algorithm is usually considered efficient if its worst-case running time, $T_{wc}(n)$ is described by the recurrence:

$$T_{wc}(n) = cT_{wc}(\lceil n/c \rceil) + O(n),$$

where c is some positive integer greater than 1. Give a closed-form expression for $T_{wc}(n)$ in Θ -notation.

Solution: This recurrence can be solved using the master theorem: using the textbook’s notation (p. 73), we have $a = b = c$. Note that $n^{\log_c c} = n$. We know $f(n)$ is $O(n)$. It could also be $\Omega(n)$, so $T_{wc}(n)$ falls under case 2 and is $O(n \log n)$.

- (2) (a) (5 points) Given a universe U with a probability distribution \Pr , and a natural number m , describe what it means for a hash function

$$h : U \longrightarrow \{0, 1, 2, \dots, m - 1\}$$

to be simple, uniform.

Solution: Let $A_i = \{k \in U : h(k) = i\}$. Simple, uniform means that $\Pr(A_i) = 1/m$ for any $i \in \{0, \dots, m - 1\}$. In words: given a random key value from the distribution on U , it is equally likely to hash to any value in $\{0, \dots, m - 1\}$ under h .

- (b) (5 points) Let $U = \{0, 1, 2, \dots, 99\}$ and let $m = 10$. Is there any probability distribution on U such that the hash function $h(x) = \lfloor (x \bmod m)/2 \rfloor$ is simple-uniform? Explain.

Solution: Notice that applying h to any number gives you an integer in the range $[0, 4]$. No matter what the distribution on U , it is impossible for a random key-value to have equal probability of hashing to anything in $\{0, \dots, m - 1\}$. For example, $\Pr(A_6) = 0 \neq 1/10$.

- (3) (a) (5 points) Consider a dictionary ADT with elements in it. Define the *rank* of a key-value k in this dictionary.

Solution: The rank of k is the number of elements in the dictionary with key-value less than or equal to k .

- (b) (5 points) Recall the query $\text{SELECT}(R, r)$, which, given a dictionary R , returns the key-value in R of rank r . Using the standard dictionary operations and SELECT , write pseudo-code which outputs the key values in the dictionary in sorted order. The code should not modify the contents of the dictionary.

Solution: Let's assume that if there is no key-value of rank r , then $\text{SELECT}(R, r)$ returns NIL.

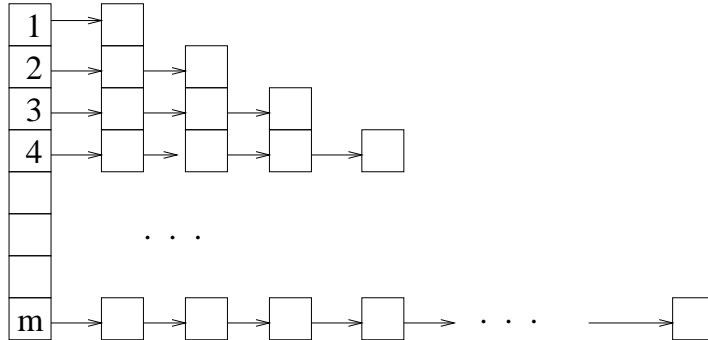
```
i ← 1
DO {
  x ← SELECT(R, i)
  IF ( x ≠ NIL ) THEN
    PRINT x
  i++
} WHILE ( x ≠ NIL )
END
```

- (c) (5 points) In class we augmented red-black trees to support SELECT . If we use red-black trees as the data structure to implement the algorithm in part (b), what is the worst-case running time of the algorithm?

Solution: In class, we showed how to support SELECT in worst-case time $O(\log n)$. The above pseudo-code would then run in time $O(n \log n)$.

- (4) Consider a hash table with chaining that has m cells numbered $\{1, 2, \dots, m\}$. For each i , cell i has a chain of i elements in it (see picture).

Cell #



Throughout this question, the sums

$$\sum_{i=1}^m i = m(m-1)/2 \quad \sum_{i=1}^m i^2 = m(m+1)(2m+1)/6$$

might be useful.

- (a) (5 points) Assume our sample space S is the set of elements in the hash table with a uniform distribution. What is the probability of each element?

Solution: The number of elements in the hash table is just $\sum_{i=1}^m i = m(m+1)/2$. Since we are using the uniform distribution, the probability of each element is just the reciprocal: $2/(m(m+1))$.

- (b) (5 points) Let $A_i \subset S$ be the event of elements that can be found (using SEARCH) in i steps. What is $\Pr(A_i)$.

Solution: The number of elements that are first in their chain is m . The number of elements that are second in their chain is $m-1$. The number of elements that are i th in their chain, is $m-i+1$. Therefore, the size of A_i is $m-i+1$. Then $\Pr(A_i)$ is just this size times the probability of each element: $(m-i+1)2/(m^2+m)$.

- (c) (5 points) What is the average-case running time of SEARCH?

Solution: This is the sum $\sum_{i=1}^m \Pr(A_i) \cdot i$, because it takes i comparisons to find an element in A_i . Expanding this out, we get:

$$\begin{aligned} T_{avg}(n) &= \sum_{i=1}^m (m-i+1)2i/(m^2+m) \\ &= 2/(m^2+m) \sum_{i=1}^m mi - i^2 + i \\ &= 2/(m^2+m) (m^2(m+1)/2 - m(m+1)(2m+1)/6 + m(m+1)/2) \\ &= m - (2m+1)/3 + 1 \\ &= (m+2)/3. \end{aligned}$$