Problem 1. Recall that our RAM model has an atomic operation $\textsc{Random}(x, y)$ which, given integers x, y, returns an integer chosen uniformly at random from $[x, y]$. Suppose that you are allowed to call the operation only with $x = 1$ and $y = 128$. Describe an algorithm to obtain a uniformly random number between 1 and 100. Your algorithm must finish in $O(1)$ expected time.

Solution. Consider the following algorithm that rejects random number out of range and regenerates a new candidate.

do
  $r \leftarrow \textsc{Random}(1, 128)$
while $r \notin [1, 100]$

We verify that this algorithm generates uniform random integer from 1 to 100.

$$\mathbb{P}(x = k) = \frac{1}{128} + \frac{28}{128} \times \frac{1}{128} + \left(\frac{28}{128}\right)^2 \times \frac{1}{128} + \ldots = \frac{1}{128}\left(\frac{1}{1 - \frac{28}{128}}\right) = \frac{1}{100}.$$

In expectation, the number of calls to $\textsc{Random}$(1,128) is

$$\mathbb{E}[\#\text{calls}] = \frac{100}{128} + \frac{28}{128} \times \frac{100}{128} \times 2 + \left(\frac{28}{128}\right)^2 \times \frac{100}{128} \times 3 + \ldots = \frac{128}{100}$$

and hence gives the desired expected time complexity $O(1)$.

Problem 2. Suppose that we enforce an even harder constraint that you are allowed to call $\textsc{Random}(x, y)$ only with $x = 0$ and $y = 1$. Describe an algorithm to generate a uniformly random number in $[1, n]$ for an arbitrary integer n. Your algorithm must finish in $O(\log n)$ expected time.

Solution.

Let k be the smallest integer satisfying $2^k \geq n$. We generate a bit string of length k by calling the procedure call k times. Viewing the string as a binary representation of a decimal integer x allows us to map the bit string from $\{0, 1\}^n$ to $[1, 2^k]$. Hence, we have obtained an algorithm for $\textsc{Random}(1, 2^k)$ that runs in $O(k)$ expected time.

Modify the algorithm from Problem 1 as follows:

do
  $r \leftarrow \textsc{Random}(1, 2^k)$
while $r \notin [1, n]$

We can verify again that the new generator is uniform, runs in $O(\log n)$ time expected as desired.

Problem 3. Consider the following algorithm to find the greatest common divisor of n and m where $n \le m$:

```
procedure GCD(n, m)
    if n = 0 then
        return m
    end if
    m ← m − n
    if n ≤ m then
        return GCD(n, m)
    else
        return GCD(m, n)
    end if
end procedure
```

Prove:

(a) The time complexity of the algorithm is $O(m)$.

(b) The time complexity of the algorithm is $\Theta(m)$.

Proof.

(a) In each recursive call, $\max\{n, m\}$ at least decreases by 1. Therefore, at most $2m$ calls is required to force $n = 0$. Each call clearly requires $O(1)$ time only.

(b) Fix $n = 1$, then the algorithm must make m calls.

□

Problem 4. Consider an input array A that has $n = 120$ elements. Suppose that we choose a number v in A uniformly at random. What is the probability that the rank of v (among all the numbers in A) fall in the range $[35, 78]$?

Solution. Let B be array A after sorting.

$$\mathbb{P}(35 \le \operatorname{rank}(v) \le 78) = \mathbb{P}(b_{35} \le v \le b_{78}) = \frac{78 - 35 + 1}{120} = \frac{11}{30}.$$

Problem 5. **(A Simpler Randomized Algorithm for k-Selection, but with a More Tedious Analysis)** In the k-selection problem, we have an array S of n distinct integers (not necessarily sorted). We would like to find the k-th smallest integer in S where $k \in [1, n]$. Here is another way of solving it using randomization. If $n = 1$, then we simply return the only element in S. For $n > 1$, we proceed as follows:

- Randomly pick an integer v in S, and obtain the rank r of v in S.
- If $r = k$, return v.
- If $r > k$, produce an array S' containing the integers of S that are smaller than v. Recurse by finding the k-th smallest in S'.
- Otherwise, produce an array S' containing the integers of S that are larger than v. Recurse by finding the $(r - k)$-th smallest in S'.

Prove that the above algorithm finishes in $O(n)$ expected time.

Proof. First, the rank can be obtained in $O(n)$ time by counting the number of element in S smaller than v. Scanning the array once again generates the array S' in $O(n)$ time.

Second, $\mathbb{P}(r = i) = 1/n$. Notice that v splits the array S into two subarrays with elements smaller than v and larger than v, respectively. In the worst case, the algorithm recurses into the larger of the two and thus size of S' is $\max\{i - 1, n - i\}$.

The expected time complexity of the algorithm is then:

$$f(n) \le \alpha n + \frac{1}{n} \sum_{i=1}^{n} f(\max\{i - 1, n - i\}) \le \alpha n + \frac{2}{n} \sum_{i=\lceil n/2 \rceil}^{n} f(i - 1).$$

We claim that this algorithm runs in $O(n)$ expected time.

We can choose β such that $f(n) \le \beta n_0$ for any $n \le 8$. Let $n_0 = 8$. Suppose that $f(n_0) \le cn$ for any $n < n_0$ and $c = \max\{\beta, 8\alpha\}$, we argue that $f(n) \le cn$.

$$\begin{aligned} f(n) &\le \alpha n + \frac{2}{n} \sum_{i=\lceil n/2 \rceil}^{n} f(i - 1) \le \alpha n + \frac{2}{n} \sum_{i=\lceil n/2 \rceil}^{n} c(i - 1) = \alpha n + \frac{2c}{n} \cdot \frac{\left(\left\lceil \frac{n}{2} \right\rceil + n - 2\right)\left(n - \left\lceil \frac{n}{2} \right\rceil + 1\right)}{2} \\ &\le \alpha n + \frac{2c}{n} \frac{(n/2 + 1 + n - 2)(n/2 + 1)}{2} \le \alpha n + \frac{2c}{n} \frac{(3n/2 - 1)(n/2 + 1)}{2} \\ &\le \alpha n + \frac{c}{n}(3n^2/4 + 3n/2 - n/2 - 1) \le \alpha n + \frac{c}{n}(3n^2/4 + n - 1) < \alpha n + 3cn/4 + c. \end{aligned}$$

We require that $\alpha n + 3cn/4 + c \le cn \Rightarrow 4\alpha n + 4c \le cn$.

As $c \ge 8\alpha, n \ge 8$, we have $4\alpha n + 4c \le \max\{8\alpha n, 8c\} \le cn$. $\square$

Problem 6. Explain how to implement the operation $x \bmod y$ in $O(1)$ time where x and y are positive integers.

Solution. We apply the basic arithmetic operations defined in the RAM model.

Clearly we can let $a \leftarrow x/y$, and $b \leftarrow x - a \cdot y$ in $O(1)$ time. Now, b should store $x \bmod y$.

Problem 7. For the k-selection problem, suppose that the input is an array of 12 elements: $(58, 23, 98, 83, 32, 24, 18, 45, 85, 91, 2, 34)$. Recall that our randomized algorithm first selects a number v and then recursively solves a subproblem. Suppose that $v = 34$ and $k = 10$. What is the size of the array for the subproblem?

Solution. $k > rank(v) = 6$. The algorithm recurses into a subproblem with size $12 - rank(v) = 6$ and elements $(58, 98, 83, 45, 85, 91)$.

Problem 8. The *median* of a set S of n elements is the $\lfloor n/2 \rfloor$ smallest element in S. Suppose that you are given a deterministic algorithm for finding the median of S (stored in an array) in $O(n)$ worst-case time. Using this algorithm as a black box, design another deterministic algorithm for solving the k-selection problem (for any $k \in [1, n]$) in $O(n)$ worst-case time.

Solution. We can always select the median as pivot for a subproblem with size at most $n/2$. Therefore, we have the following recursive formula for time complexity:

$$T(1) = 1, T(n) \leq T(n/2) + O(n),$$

which evaluates to $O(n)$.

Problem 9. Let S be a set of n distinct integers, and k_1, k_2 be arbitrary integers satisfying $1 \leq k_1 \leq k_2 \leq n$. Suppose that S is given in an array. Give an $O(n)$ expected time algorithm to report all the integers whose ranks in S are in the range $[k_1, k_2]$. Recall that the rank of an integer v in S equals the number of integers in S that are at most v.

Solution. Apply the $O(n)$ expected time k-selection algorithm twice and obtain v_1, v_2 such that $rank(v_1) = k_1$ and $rank(v_2) = k_2$. Now we scan the array again and output the elements that takes values v with $v_1 \leq v \leq v_2$.

Problem 10. We are given an array that stores a set S of n distinct positive integers. Set $W = \sum_{e \in S} e$. Describe an algorithm to find the element $e^* \in S$ that makes both of the following hold:

- $\sum_{e<e^*} e < W/2$
- $\sum_{e>e^*} e \le W/2$

Your algorithm should finish in $O(n)$ expected time.
(Hint: First convince yourself that such e^* is unique, and then adapt the k-selection algorithm).

Solution. Such e^* must exist, and is unique. By construction, let $e_1 \in S$ such that $\sum_{e<e_1} e < W/2$ and $\sum_{e \le e_1} e \ge W/2$. Hence, $\sum_{e>e_1} e \le \sum_{e \in S} e - W/2 = W/2$. Clearly, e_1 is the largest element in S satisfying the first condition. Suppose we have $e_2 < e_1$ that also satisfies the first requirement, we have $\sum_{e \le e_2} e < W/2$ implied by our construction and thus $\sum_{e>e_2} e > W/2$, which is a violation of condition 2. We conclude that e_1 is the desired element.

The algorithm uses the following procedure call $\textsc{Find}(S, W/2)$, and aims to find the element e^* which has $\sum_{e<e^*} e$ as close as and smaller than $W/2$.

procedure $\textsc{Find}(S, target)$
  if $|S| = 1$ **then**
    return $S[1]$
  end if
  Randomly pick an integer v in S, and obtain $s_1 = \sum_{e<v} e, s_2 = \sum_{e \le v} e$
  if $s_1 < target$ **then**
    if $s_2 \ge target$ **then**
      return v
    else
      Produce an array S' containing the integers of S that are larger than v
      return $\textsc{Find}(S', target - s)$
    end if
  else
    Produce an array S' containing the integers of S that are smaller than v
    return $\textsc{Find}(S', target)$
  end if
end procedure

Similar to the analysis of the k-selection algorithm we obtain the following recursive formula for time complexity,

$$f(n) \le \alpha n + \frac{1}{n} \sum_{i=1}^{n} f(\max\{i-1, n-i\}),$$

which resolves to $O(n)$ expected time.