# CIS507: Design & Analysis of Algorithms
## *Homework 1 (with answers), Spring 2014*

March 5th, 2014

**Student Name: Yanan Xiao**

**Student ID: 131102019**

**Q1.** Give asymptotic upper bounds for $T(n)$ for each of the following recurrences (use the $O$-notation). Assume that $T(n)$ is a non-negative constant for $n$ sufficiently large (in terms of $\alpha$). Make your bounds as tight as possible, and justify your answers.

1. **(1 point)** $T(n) = n^{1-\alpha} \cdot T(n^\alpha) + \Theta(n)$, for a constant $\alpha \in (0, 1)$.
2. **(1 points)** $T(n) = T(n-1) + T(\alpha \cdot n) + 1$, for a constant $\alpha \in (0, 1)$.

**ANSWER:**

1. $$T(n) = n^{1-\alpha} \cdot T(n^\alpha) + \Theta(n) \Leftrightarrow \frac{T(n)}{n} = \frac{T(n^\alpha)}{n^\alpha} + \Theta(1)$$

   Let
   $$\frac{T(n)}{n} = P(n) \Rightarrow \frac{T(n)}{n} = \frac{T(n^\alpha)}{n^\alpha} + \Theta(1) \Leftrightarrow P(n) = P(n^\alpha) + \Theta(1)$$

   Let
   $$n = 2^m \Leftrightarrow m = \log n \Rightarrow S(2^m) = S(2^{\alpha m}) + \Theta(1)$$

   Let
   $$Q(m) = P(2^m) \Rightarrow Q(m) = Q(\alpha m) + \Theta(1)$$

   By Master Theorem, we can get
   $$m^{\log_b a} = m^{\log_{\frac{1}{\alpha}} 1} = m^0 = 1 \Rightarrow Q(m) = \Theta(\log m) \Rightarrow P(n) = \Theta(\log \log n)$$

   But
   $$T(n) = nP(n) \Rightarrow T(n) = \Theta(n \log \log n) = O(n \log \log n)$$

2. From the question description we can see that $T(n)$ is sufficiently large in terms of $\alpha$.
   To be specific, let's assume $\alpha = 0.5$, then we can rewrite the recurrence equation as $T(n) = T(n-1) + T(0.5n) + 1$. By using tree induction it is not hard to guess $T(n) = O(n \log n)$. The proof is as follows.

**Q2. (2 points)** Consider the following problem called MAXCUT: given an undirected graph $G = (V, E)$ with non-negative edge weights $w_e$ for $e \in E$, find a partition $(S, V \setminus S)$ of the vertices that maximizes the total weight of the edges crossing the cut, that is, $\sum_{e \in \delta(S)} w_e$, where $\delta(S)$ is the set of edges that have one end-point in $S$ and another in $V \setminus S$.

Consider the following randomized algorithm: Select a subset $S$ by picking each vertex in $V$ independently with probability $\frac{1}{2}$. Show that the expected wight of the edges in the cut $(S, V \setminus S)$ is a factor of $\frac{1}{2}$ of the total weight, that is:

$$\mathbb{E}\left[\sum_{e \in \delta(S)} w_e\right] = \frac{1}{2} \sum_{e \in E} w_e$$

(*Hint:* use an indicator random variable for each edge.)
**ANSWER:** Let's define an indicator random variable $P_e$ where

$$P_e = \begin{cases} 1 & \textit{the edge crosses the cut} \\ 0 & \textit{the edge does not cross the cut} \end{cases} \tag{1}$$

Since each vertex is picked up independently with a 0.5 probability, therefore the probability for a vertex's both ends are in the same set is 0.5. Thus we have $\mathbb{E}[P_e] = 0.5$.

$$\sum_{e \in \delta(S)} w_e = \sum_{e \in E} w_e P_e$$

Take the expectation of both sides, we have

$$E[\sum_{e \in \delta(S)} w_e] = E[\sum_{e \in E} w_e P_e]$$

With the linearity of expectation it's not hard to get

$$E[\sum_{e \in \delta(S)} w_e] = \sum_{e \in E} w_e \mathbb{E}[P_e] = \frac{1}{2} \sum_{e \in E} w_e$$

**Q3.** Suppose that we would like to analyze the change in price for a given stock. We observe the different prices over a period of $n$ days. Let $A[i]$ be the observed price in day $i$. We would like to compute:

(I) the smallest absolute price difference: $\min_{1 \leq i,j \leq n, i \neq j} |A[i] - A[j]|$;

(II) the largest absolute price difference: $\max_{1\le i,j\le n} |A[i]-A[j]|$;

(III) the average absolute price difference: $\frac{1}{n(n-1)} \sum_{1\le i,j\le n} |A[i]-A[j]|$;

(IV) the median absolute price difference: $\text{median}(\{|A[i]-A[j]| : 1 \le i,j \le n\})$.

(i) **(1 point)** give an $O(n^2)$ deterministic algorithm for computing (I), (II), (III) and (IV);

(ii) **(1 point)** give an $O(n \log n)$ deterministic algorithm for computing (I);

(iii) **(1 point)** give an $O(n)$ deterministic algorithm for computing (II);

(iv) **(1 point)** give an $O(n \log n)$ deterministic algorithm for computing (III);

(v) **(1 point)** give a randomized algorithm with $O(n^2)$ expected running time for computing (IV).

Implement the four algorithms in (ii), (iii), (iv) and (v). For testing purposes, your program should accept as an input a file "test.in", containing $n$, followed by the set of $n$ numbers (1 per line). It should output the four values described in (I), (II), (III), and (IV).

**ANSWER:** In the following solutions we will use two auxiliary algorithms. One is standard merge sort, the other is a randomized selection algorithm.

**Input:** An array $A$ of length $n$
**Output:** The smallest absolute difference

**for** $i = 1$ to $n$ **do**
  **for** $j = i+1$ to $n$ **do**
    $B[k] = |A[i]-A[j]|$
$min = B[1]$
**for** $i = 1$ to $len(B)$ **do**
  **if** $B[i] < min$ **then**
    $min = B[i]$
return $min$

Figure 1: Deterministic Algo for Smallest Abs

**Q4. (4 points)** Implement a *perfect* hash table, where keys are decimal numbers, each having at most 10 digits. For both hash levels, use the class of universal hash functions of the dot-product form: if the hash table size is a prime $m$, pick a random sequence $\mathbf{a} := \langle a_0, a_1, \ldots, a_9 \rangle$, where each $a_i \in \{0, 1 \ldots, m-1\}$; given a key $k$, decompose it into a sequence of decimal digits $\mathbf{k} := \langle k_0, k_1, \ldots, k_9 \rangle$, then use hash functions of the form $h_{\mathbf{a}}(k) = (\sum_{r=0}^{9} a_i k_i) \bmod m$. Your table should have no collisions, and and uses at most $8n$ table entries, in total. (*Hint:* use the fact that for any positive integer $n$, there is at least one prime between $n$ and $2n$.)

**Input:** An array $A$ of length $n$
**Output:** The largest absolute difference
  **for** $i = 1$ to $n$ **do**
    **for** $j = i + 1$ to $n$ **do**
      $B[k] = |A[i] - A[j]|$
  $max = B[1]$
  **for** $i = 1$ to $len(B)$ **do**
    **if** $B[i] > max$ **then**
      $max = B[i]$
  return $max$

Figure 2: Deterministic Algo for Largest Abs

**Input:** An array $A$ of length $n$
**Output:** The average absolute difference
  $C = 0$
  **for** $i = 1$ to $n$ **do**
    **for** $j = i + 1$ to $n$ **do**
      $B[k] = |A[i] - A[j]|$
      $C = C + B[k]$
  return $\frac{C}{n(n-1)}$

Figure 3: Deterministic Algo for Avg Abs

**Input:** An array $A$ of length $n$
**Output:** The median absolute difference
  **for** $i = 1$ to $n$ **do**
    **for** $j = i + 1$ to $n$ **do**
      $B[k] = |A[i] - A[j]|$
  $C =$ Merge-sort$(B)$
  return $C[mid]$

Figure 4: Deterministic Algo for Median Abs

**Input:** An array $A$ of length $n$
**Output:** The smallest absolute difference
  $B =$ Merge-sort$(A)$
  $c = B[2] - B[1]$
  **for** $i = 1$ to $n - 1$ **do**
    $d = B[i + 1] - B[i]$
    **if** $d < c$ **then**
      $c = d$
  return $c$

Figure 5: Deterministic Algo for Smallest Abs with $O(n \log n)$

**Input:** An array $A$ of length $n$
**Output:** The largest absolute difference
$min = A[1], max = A[1]$
**for** $i = 1$ to $n$ **do**
  **if** $A[i] < min$ **then**
    $min = A[i]$
  **if** $A[i] > max$ **then**
    $max = A[i]$
return $max - min$

Figure 6: Deterministic Algo for Largest Abs with $O(n)$

**Input:** An array $A$ of length $n$
**Output:** The average absolute difference
$B =$ Merge-sort $(A)$
$sum = 0$
**for** $i = 1$ to $n$ **do**
  $sum = sum + B[i] * (2 * i - len(B) + 1)$
return $\frac{sum}{n(n-1)}$

Figure 7: Deterministic Algo for Avg Abs with $O(n \log n)$

**Input:** An array $A$ of length $n$
**Output:** The median absolute difference
**for** $i = 1$ to $n$ **do**
  **for** $j = i + 1$ to $n$ **do**
    $B[k] = |A[i] - A[j]|$
$C$= Randomized-select $(A, 1, n, \lfloor n/2 \rfloor)$
return $C$

Figure 8: Randomized Algo for Median Abs with $O(n^2)$

For testing purposes, your program should accept as an input a file "test.in" containing the number of keys $n$, followed by the set of keys to be hashed (1 per line). It should output in another file "test.out", the following lines: the first line (call it line 0) contains the values chosen for the first-level hash function in the following order (separated by spaces): $m, a_0, a_1, \ldots, a_r$; then for $i = 1, \ldots, m$, the $i$th line contains the values corresponding to the second level-hash function chosen at the $i$th row in the first level table (again in the order $m(i), a_0(i), a_1(i), \ldots, a_r(i)$; output "0 0" if that row is empty). Following this, the file should contain triples (one per line): $(k, h(k), h_i(k))$, where $k$ is the key, $i = h(k)$ is the index in the first-level hash table, $h_i(k)$ is the index in the second level hash table.