ShanghaiTech University

CS101 Algorithms and Data Structures
Fall 2024

Homework 3

Due date: October 23, 2024, at 23:59

1. Please write your solutions in English.
2. Submit your solutions to Gradescope.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Gradescope account settings.
4. If you want to submit a handwritten version, scan it clearly. `CamScanner` is recommended.
5. We recommend you to write in LaTeX.
6. When submitting, match your solutions to the problems correctly.
7. No late submission will be accepted.
8. Violations to any of the above may result in zero points.

Important Notes:

1. Some problems in this homework requires you to design divide-and-conquer algorithm. When grading these problems, we will put more emphasis on how you reduce a problem to a smaller size problem and how to combine their solutions with divide-and-conquer strategy.

2. There are mainly 3 kinds of writing problems that may be involved in this assignment (and following assignments). We recommend you to know what each part **should** include:

    1. Describe/Design an algorithm that...: Clear description of your algorithm design in natural language is needed. If you want to present it into pseudocode, you'd better take it seriously on your boundary cases and supplement it with meaningful, clear variable names. Unless the problem allows, you should give a complete algorithm instead of fragmented words incremented by some other algorithm.

    2. Justify your algorithm/Justify its correctness/...: You should give out a proof of why your algorithm is correct. The requirements may not be so strict, but please at least clarify every step of your proof and make sure they are reasonable. You can give out an answer or some propositions and then make **reliable and credible** explanations, and we encourage you to do that on suitable cases (e.g. using induction).

    3. Analyze the time complexity/Show its time complexity is ...: You should give out a exact time complexity and some reason to justify it (e.g. Master Theorem/Recurrence relation/...). We won't restrict too much on O and Θ. However, your time complexity bound should be **tight**. For example, we won't make it correct when you say something is $O(n^2)$ with its worst case running on $\Theta(n \log n)$ (Even if it is logically correct).

3. Your answer for these problems is not allowed to include real C or C++ code.

4. In your description of algorithm design, you should describe each step of your algorithm clearly.

5. You are encouraged to write pseudocode to facilitate explaining your algorithm design, though this is not mandatory. However, we recommend you not to write pseudocode, if you are using handwriting to finish your homework. If you choose to write pseudocode, please give some additional descriptions to make your pseudocode intelligible.

6. You are recommended to finish the algorithm design part of this homework with LaTeX. If you insist on using handwriting to finish it, your explanation should be at least tidy and readable (can be specified) otherwise your assignments will be considered as unspecified.

7. Please **do not** just hand in a picture of answer-sheet with several answers only. For both of your and our convenience, please at least upload a PDF assignment file (In fact, you don't need to cut down any pages because we will filter it for you).

1. (0 points) Binary Search Example

Design an algorithm to find the index of an element x in a sorted array a of n elements.

Solution:

We will show how you are supposed to finish the 3 kinds of problems mentioned last page.

Algorithm Design: We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with the middle element, return the middle index.
3. Otherwise x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Otherwise (x is smaller) recur for the left half.

Pseudocode(Optional):

```
function BINARYSEARCH(a, value, left, right)
    if right < left then
        return not found
    end if
    mid ← $\lfloor (right - left)/2 \rfloor + left$
    if a[mid] = value then
        return mid
    end if
    if value < a[mid] then
        return binarySearch(a, value, left, mid-1)
    else
        return binarySearch(a, value, mid+1, right)
    end if
end function
```

Proof of Correctness: If x happens to be the middle element, we will find it in the first step. Otherwise, if x is greater than the middle element, then all the element in the left half subarray is less than x since the original array has already been sorted, so we just need to look for x in the right half subarray. Similarly, if x is less than the middle element, then all the element in the right subarray is greater than x, so we just need to look for x in the front list. If we still can't find x in a recursive call where $left = right$, which indicates that x is not in a, we will return *not found* in the next recursive call.

Time Complexity Analysis: During each recursion, the calculation of *mid* and comparison can be done in constant time, which is $O(1)$. We ignore half of the elements after each comparison, thus we need $O(\log n)$ recursions.

$$T(n) = T(n/2) + O(1)$$

Therefore, by the Master Theorem $\log_b a = 0 = d$, so $T(n) = O(\log n)$.

2. (15 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 1 point if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)	(d)	(e)
AB	AC	D	ABCD	AC

(a) (3') Which of the following sorting algorithm(s) is(are) stable?

- **A. Insertion-Sort**
- **B. Merge-Sort**
- C. Quick-Sort (picking a random element as pivot)
- D. None of the above

Solution:

C. Not stable even always picking the first element as pivot.

(b) (3') Which of the following statements is **NOT** true?

- **A. The worst case time complexity of merge-sort is $O(n^2)$.**
- B. Given 2 sorted lists of size m and n respectively, and we want to merge them to one sorted list by merge-sort. Then in the worst case, we need $m + n - 1$ comparisons.
- **C. The time complexity of quick-sort, compared with merge-sort, is less affected by the initial order of the input array.**
- D. Traditional implementations of merge-sort need $\Theta(n \log n)$ time when the input sequence is sorted or totally reversely sorted, but it is possible to make it $\Theta(n)$ on such input while still $\Theta(n \log n)$ on the average case.

Solution:

C. Quicksort: In its worst case, quicksort can degrade to $O(n^2)$ time complexity. Mergesort: Mergesort always has a time complexity of $O(n \log n)$, regardless of the initial order of the input array.

D. Just run a pass to identify such situations and treat them specially.

(c) (3') You have to sort n data with very limited extra memory ($\Theta(1)$ extra memory every case). Choose sort algorithm(s) which is(are) **not** appropriate.

Hint: Note that too much recursion will take use of huge stack memory.

A. Merge-sort.
B. Insertion-sort.
C. Bubble-sort.
D. Quick-sort.

Solution:

A. Merge sort takes $\Theta(n)$ extra space.
There are actually some algorithms called "in-place merge sort", but when we refer to merge sort in CS101, we mean the algorithm we taught in class, which is not in-place.

D. Quick sort takes average-case $\Theta(logn)$ but worst-case $\Theta(n)$ extra space.
However, if tail recursion optimization is applied, the worst-case space complexity can be reduced to $\Theta(\log n)$, but when we refer to quick sort in CS101, we mean the algorithm we taught in class, which is with worst-case $\Theta(n)$ space complexity. Tail recursion optimization reference in CS101 WeChat official account: Tail recursion optimization

(d) (3') Which of the following implementations of quick-sort may improve the average **behavior** (not only improvement on time complexity) of trivial quick-sort?(Compared to always picking the first element as pivot and using no optimization, data are uniformly distributed.)

A. Always picking the last element.
B. When partitioning the subarray $\langle a_l, \cdots, a_r \rangle$ (assuming $r - l \geqslant 2$), choose the median of $\{a_x, a_y, a_z\}$ as the pivot, where x, y, z are three different indices chosen randomly from $\{l, l+1, \cdots, r\}$.
C. When partitioning the subarray $\langle a_l, \cdots, a_r \rangle$ (assuming $r - l \geqslant 2$), we first calculate $q = \frac{1}{2}(a_{\max} + a_{\min})$ where $a_{\max}$ and $a_{\min}$ are the maximum and minimum values in the current subarray respectively. Then we traverse the whole subarray to find a_m $s.t.$ $|a_m - q| = \min_{i=l}^{r} |a_i - q|$ and choose a_m as the pivot.
D. When the disordered sub-array is short enough, use insertion-sort instead of quick-sort.

Solution:

A. Choosing the last element (or even randomly pick one) will not affect the average behavior of the quick sort since data are uniformly distributed.

B.C. All of these tricks may improve the average behavior to a certain degree by reducing the probability of encountering the worst case or simplifying the task under certain circumstances, but note that the time complexity of the algorithm is still $\Theta\left(n^2\right)$ in the worst case.

D. The method cuts down the height of recursion tree. In other words, it turns the recurrence relation $T(n) = 2(T/n) + \Theta(n), T(1) = 1$ into $T(n) = 2T(n/2) + \Theta(n), T(c) = C(c \leq L)$, where L, c, C are constant. The magnitude of time complexity won't change but the constant item (e.g. the stack cost) will be reduced.

(e) (3') Which of the following statements is true?

A. If $T(n) = 2T(\frac{n}{2}) + O(\sqrt{n})$ with $T(0) = 0$ and $T(1) = 1$, then $T(n) = \Theta(n)$.

B. If $T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + \Theta(n)$ with $T(0) = 0$ and $T(1) = 1$, then $T(n) = O(n \log n)$.

C. If $T(n) = 4T(\frac{n}{2}) + O(n^2)$ with $T(0) = 0$ and $T(1) = 1$, then $T(n) = \Theta(n^2 \log n)$.

D. If the run-time $T(n)$ of a divide-and-conquer algorithm satisfies $T(n) = aT(\frac{n}{b}) + f(n)$ with $T(0) = 0$ and $T(1) = 1$, we may deduce that the run-time for dividing the original problem into a subproblems of size $\frac{n}{b}$ is $f(n)$.

Solution:

A. By Master Theorem: $\log_b a = 1 > d = \frac{1}{2}$, thus $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$.

B. The height of the recursive tree is $O(log_{\frac{10}{9}} n)$, and the time complexity of each layer is $\theta(n)$, thus the total time complexity is $O(n \log n)$.

C. $\Theta(n) \subseteq O(n^2)$ and when $T(n) = 4T(\frac{n}{2}) + \Theta(n)$, $T(n) = O(n^2)$ not $\Theta(n^2 \log n)$

D. $f(n)$ is actually the run-time of dividing the original problem into several sub-problems plus that of merging solutions of sub-problems into the overall one.

3. (15 points) Counting Jewelry Pairs

Astra opens up a jewelry store to sold the jewelry. Each jewelry has a distinct positive "beauty value" a_i, where i means it's the i-th jewelry Astra got. You are asked to help Astra to solve the problems for her customs.

(a) Breach wants to buy a pair of jewelry, where jewelry found earlier should be more beautiful i.e. have a bigger beauty value. In other words, the pair contains 2 jewelry i and j, where $i < j, a_j < a_i$.

Astra needs to count the number of jewelry pairs satisfying Breach's requirement.

i. (1') Recall what is told in class. This problem is equivalent to

○ Quick Sort ○ Merge Sort [✓ **Counting Inversions**] ○ Binary Search

ii. (2') Briefly write how to solve this problem using divide and conquer.

Solution:

1. **Base Case:** If the array has at most one element, return zero. An array with zero or one element cannot have any inversions.
2. **Divide:** Split the array into two halves. Denote the left half as L and the right as R.

$$L = \{a_1, a_2, \ldots, a_{\lfloor n/2 \rfloor}\}, \quad R = \{a_{\lfloor n/2 \rfloor + 1}, a_{\lfloor n/2 \rfloor + 2}, \ldots, a_n\}$$

3. **Conquer:**
   - Recursively count the inversions in the left half L, denoted as $\text{Inv}(L)$.
   - Recursively count the inversions in the right half R, denoted as $\text{Inv}(R)$.
4. **Combine:**

   (a) Combine the results by counting the number of cross-inversions, i.e., the number of inversions where an element in the left half is greater than an element in the right half. Let this count be $\text{CrossInv}(L, R)$.

   (b) For cross-inversions, merge the two halves while counting inversions:

   i. Initialize two pointers, i for the left half L starting at the beginning and j for the right half R also starting at the beginning. Initialize a count variable for cross-inversions.

   ii. When merge the two halves:

   A. If $L[i] \leq R[j]$, copy $L[i]$ to the merged array and increment i.

   B. If $L[i] > R[j]$, it means all the elements from $L[i]$ to the end of the left part can form an inversion with $R[j]$. Add the number of such elements (which is $|L| - i + 1$) to the cross-inversion count and copy $R[j]$ to the merged array and increment j.

iii. Continue this process until all elements in the two halves are merged.

(c) The total number of inversions in the array is the sum of inversions in the left half, inversions in the right half, and the cross-inversions:

$$\text{TotalInv}(a) = \text{Inv}(L) + \text{Inv}(R) + \text{CrossInv}(L, R)$$

In the two questions below, you are allowed to describe your algorithm by modifying parts of the algorithm in $(a).ii$.

(b) Chamber wants to buy a pair of jewelry too. His requirement is precise: He wants that the earlier found jewelry is "quadratic more beautiful" than the other one. In other words, for two jewelry i and $j(i < j)$, Chamber wants $a_i > \alpha a_j^2 + \beta a_j + \gamma$, where a_i, a_j is the "beauty value" of i/j-th jewelry and $\alpha, \beta > 0$.
Astra needs to count the number of jewelry pairs satisfying Chambers's requirement.
You should :

i. (4') Describe your algorithm, either by modifying or rewriting one.

Solution:
Note that $a_i > \alpha a_j^2 + \beta a_j + \gamma$ can be rewritten as $a_i > f(a_j)$.
Modify the "Merge" part of the algorithm: when it comes to the part of counting "CrossInv(L, R)", do counting and sort separately.
Note that L and R are sorted already. In the counting part:

1. If $L[i] \le \alpha R[j]^2 + \beta R[j] + \gamma$, then increment i.
2. If $L[i] > \alpha R[j]^2 + \beta R[j] + \gamma$, it means all the elements from $L[i]$ to the end of left part are "quadratic more beautiful" than $R[j]$. Add the number of such elements (which is $|L| - i + 1$) to the cross-inversion count and increment j.

The merge-sort part is the same as the original one.

ii. (2') Justify why your algorithm is correct and analyze the time complexity.

Solution: Note that $a_i > \alpha a_j^2 + \beta a_j + \gamma$ can be rewritten as $a_i > f(a_j), f(x) = \alpha x^2 + \beta x + \gamma$.
$f(x)$ is increasing on $[0, +\infty)$ (since $\alpha, \beta > 0$). When merging, the left part L and the right part R is sorted separately. So if $L[i] > f(R[j])$, then $\forall k \le i, L[k] \le L[i] > f(R[j])$ i.e. $L[k]$ is "quadratic more beautiful" than $R[j]$.
And due to the correctness of "counting inversions algorithm",every time j increments, the pair of (i, j) satisfies that the i is the minimal suitable index i.e. $\forall k < i, L[k] \le f(R[j])$, which shows the correctness of this modified algorithm.
The modified part is the comparison method, the recurrence function is still

$T(n) = 2T(n/2) + \Theta(n)$ i.e. $T(n) = \Theta(n \log n)$.
To generalize, denote A as the set of $\{a_i\}$, the algorithm can be used solve the problems of counting legal (i, j), where $a_i > f(a_j)$, f is a function monotonously increasing on A. $(f(a_i) > f(a_j) \Leftrightarrow a_i > a_j)$ with $\Theta(n \log n)$ time complexity.

(c) Cypher also wants to buy a pair of jewelry, too. Cypher wants to find the relation of "beauty value" between different jewelry. He wants that the difference of when the jewelry was found and the difference of beauty value are closed. In other words, for two jewelry i and $j(i < j)$, Cypher wants $a_i - a_j > \alpha|i - j|$, where a_i, a_j is the "beauty value" of i/j-th jewelry and $\alpha > 0$.

Astra needs to count the number of jewelry pairs satisfying Cypher's requirement.

You should :

i. (4') Describe your algorithm, either by modifying or rewriting one.

Solution:
Note that $|i - j| = j - i$ since $j > i$. Rewrite the inequality: $a_i + \alpha i > a_j + \alpha j$.
Denote $b_i = a_i + \alpha i$, which can be obtained in linear time.
Then the requirement becomes: $i < j$ and $b_i > b_j$. We use the algorithm of original counting inversions problem to solve it.

ii. (2') Justify why your algorithm is correct and analyze the time complexity.

Solution: Each pair of inversions in $\{b_i\}$ will correspond to the original problem, since $b_i > b_j \Leftrightarrow a_i + \alpha i > a_j + \alpha j \Leftrightarrow a_i - a_j > \alpha|i - j|$.
Note that the algorithm can be divided into 2 parts: Preprocessing (Getting $\{b_i\}$) part and the counting part.
The counting part costs $\Theta(n \log n)$ and the preprocessing part costs $\Theta(n)$ time because it only need one traverse.

4. (10 points) Similarity Test

Maddelena has drawn a lot of paintings. She wants to organize her work. To finish her job better, she decides to determine whether a subset of similar paintings exists. Every paint p has its own feature λ_p.

In this problem, we specify some notations: $\mathcal{P}$ is the set of paintings. Without giving rise to ambiguity, λ_p can be referred to the feature of a painting p. $\mathcal{F} = \{\lambda_p | p \in \mathcal{P}\}$ denotes the set of features. n can be refereed to the number of paintings i.e. $n = |\mathcal{P}|$.

Note that $\mathcal{F}$ is not partially ordered i.e. you can't make comparisons for any $p, q \in \mathcal{P}$ so that you cannot sort $\mathcal{F}$ or give out inferences like $\lambda_p < \lambda_q$ or $\lambda_q < \lambda_p$. There exists an equivalence relation on $\mathcal{F}$, as $\lambda_p = \lambda_q$ indicates p and q has similar features.

The master feature is defined as the feature it is similar to **over** half of the paintings. In other words, the "master" feature is the feature$\lambda_0 \in \mathcal{F}$ s.t. $|\{p \in \mathcal{P} | \lambda_p = \lambda_0\}| > \frac{n}{2}$.

For example, 2 sets of paintings are shown: The first set of paintings has a "master" feature $\lambda_0 = \alpha$ since there exists over a half (4: a,b,e,g) paintings share this feature. The second set of paintings does not have a "master" feature because there exists no such feature that over half paintings share that.

Painting	a	b	c	d	e	f	g
Feature	α	α	β	ω	α	β	α

Table 1: A "master" feature $\lambda_0 = \alpha$.

Painting	A	B	C	D	E
Feature	α	β	ω	β	α

Table 2: No "master" feature.

(a) Maddelena wants to find a "master" feature first. She asks you to help her with those problems:

i. (3') Design a divide-and-conquer algorithm whose worst case takes $\Theta(n \log n)$ time.
Note: Of course, you can index them, but we recommend you not to.
Hint: Recall merge sort, think out the relation between the "master" features of the 2 subsets after dividing and the "master" feature of the original set. (If they have)

Solution:
After dividing, at least one of the "master" features of the 2 subsets is the "master" feature of the original set.

1. If the remaining set $\mathcal{S}$ has only one element, return it.
2. Otherwise, denote the recent set of painting as $\mathcal{S}$. Randomly divide it into disjoint halves $\mathcal{S}_\infty$ and $\mathcal{S}_\in$, where $\mathcal{S} = \mathcal{S}_\infty \cup \mathcal{S}_\in$. Recursively find the "master" features of the 2 subsets λ_1 and λ_2. (If exists) If $\lambda_1 = \lambda_2$ (including two failures), return it.
3. Traverse the whole set$\mathcal{S}$, and calculate the $n_1 = |\{s \in \mathcal{S} | \lambda_s = \lambda_1\}|$ and $n_2 = |\{s \in \mathcal{S} | \lambda_s = \lambda_2\}|$ (Compare each paint in the set to the two features).
4. If neither of them is a "master" feature i.e. $n_1, n_2 \leq \frac{|\mathcal{S}|}{2}$, return failure. Otherwise return the "master" feature of $\mathcal{S}$ i.e. return λ_i that $n_i > \frac{|\mathcal{S}|}{2}$.

The algorithm will output either a failure or a result. If it outputs a failure, there is no master feature in those paintings. Otherwise, the result is the master feature.

ii. (2') Justify it by proving its correctness and show its time complexity is $\Theta(n \log n)$.

Solution:
Consider about dividing it into disjoint halves $\mathcal{S}_\infty$ and $\mathcal{S}_\in$, where $\mathcal{S} = \mathcal{S}_\infty \cup \mathcal{S}_\in$.
If there exists a "master" feature of $\mathcal{S}$, then consider the appearance time of it in halves: $n_1 = |\{s \in \mathcal{S}_1 | \lambda_s = \lambda_0\}|$ and $n_2 = |\{s \in \mathcal{S}_2 | \lambda_s = \lambda_0\}|$. If λ_0 is neither the "master" feature of $\mathcal{S}_1$ and the "master" feature of $\mathcal{S}_2$. Then $n_1 + n_2 \leq \frac{|\mathcal{S}_1|}{2} + \frac{|\mathcal{S}_2|}{2} = \frac{|\mathcal{S}|}{2}$, which contradicts to that λ_0 is the "master" feature of $\mathcal{S}$.
The recurrence relation of the time complexity is $T(n) = 2T(n/2) + \Theta(n)$, since traverse the whole set $\mathcal{S}$ take $\Theta(|\mathcal{S}|)$ time. By Master Theorem, $T(n) = \Theta(n \log n)$.

(b) Maddelena finds $\Theta(n \log n)$ still not fast enough since she has so many paintings. She wants to develop a new algorithm that takes $o(n \log n)$ time.
In this problem, you can always assume that the total remaining number is even. In other words, you don't need to consider how to deal with the one superfluous after dividing.

Hint: Recall binary search, what if we give up at least half every recurrence? Match those by pairs and do differently based on whether they are similar.

i. (2') Design a divide-and-conquer algorithm that takes $o(n \log n)$ time.

Solution:

1. Combine the paintings pair by pair.
2. Compare whether 2 features are the similar. If they are similar, we leave one of them. Otherwise, we drop them all.
3. After dividing for several times, it will leave one feature. Traverse the whole set and check whether it is the "master" feature.

ii. (2') Justify its correctness.

Solution: We show that:

1. Every recurrence will leave at most $\frac{n}{2}$ paintings,
2. The remaining set $\mathcal{R}$ contains a master feature λ' if the original set contains a master feature λ and $\lambda = \lambda'$.

Based on those propositions, we can cut down the scales of the problems. After finite steps, the algorithm will end. Then we prove the propositions above.

1. We find that during one recurrence if the 2 paintings in the pair are similar ones, we will drop 1, and if they are dissimilar, we will drop 2. And there exists $\frac{n}{2}$ pairs. Therefore, every recurrence will drop at least $\frac{n}{2}$ paintings.
2. If the original set contains a master feature λ, consider the dividing the set into 2 halves: N_1 Pairs of similar features and N_2 pairs of dissimilar features ($N_1 + N_2 = \frac{n}{2}$). The remaining set $|\mathcal{R}| = N_1$.
3. If there exists n_1 pairs of 2 master features (which is(are) remaining), and n_2 master features are dropped (in other words, in those N_2 pairs), we know that $2n_1 + n_2 > \frac{N}{2} = N_1 + N_2$. By definition, every dropped master feature should be in distinct pairs. So we get $N_2 \geq n_2 \Rightarrow N_2 - n_2 \geq 0$. So $2n_1 > N_1 + N_2 - n_2 \geq N_1 = |\mathcal{R}|$. That leads to that the remaining set $\mathcal{R}$ contains the same master feature as the one in the original set.

iii. (1') Analyze the time complexity of the algorithm.

Solution: The recurrence relation is $T(n) = T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n)$.

5. (16 points) Calculating with dividing

In this problem, we will give out some recurrence relation of the run-time $T_i(n)$. We want you to find the asymptotic order of $T_i(n)$ i.e. find a function $f(n)$ s.t. $T_i(n) = \Theta(f(n))$, depending on the recurrence relation given. You should justify your answer correctly.

As a reminder, to receive the points, please keep in mind that:

1. Use $T_i(n)$ to represent the run-time in i-th problem.
2. Make sure your upper bound for $T_i(n)$ is tight enough.
3. Make sure you have a reasonable explanation other than just giving out an assumption and verifying it by your sense.
4. For your convenience, you can always assume $T_i(n)$ is increasing. (May Not Strictly)

Hint: Guessing the upper bound of $T_i(n)$ and then proof it rigorously (e.g. using mathematical induction) is acceptable. However, simply plugging your guessed upper bound into the recurrence relation and verifying whether your guessed upper bound makes sense or not will make you lose points for reasoning. If you find it hard to prove something straight, try induction instead.

In each subproblem, you may ignore any issue arising from whether a number is an integer as well as assuming $T_i(0) = 0$ and $T_i(1) = 1$. You can make use of the Master Theorem, Recursion Tree, or other reasonable approaches to solve the following recurrence relations.

(a) (4') $T_1(n) = \begin{cases} \Theta(1), & 1 \le n \le k \\ T_1(k) + T_1(n-k) + \Theta(n), & n > k \end{cases}$. ($k$ is a constant.)

Solution:
Assume $n = pk + q, 0 \le q < k$, then $T_1(n) = T_1(n-k) + T_1(k) + \Theta(n)$.
$T_1(k) = \Theta(1)$ given above, so $T_1(n) = T_1(n-k) + \Theta(n)$.
Then $T_1(n) = \Theta(\sum_{i=0}^{p}(ki+q)) + T_1(q) = \Theta(pq + \frac{p(p+1)}{2}k) = \Theta(\frac{1}{k}n^2) = \Theta(n^2)$
$T_1(n) = \Theta(n^2)$.
Another way to prove it's tight:
$T_1(pk) \le T_1(n) \le T_1((p+1)k)$, so $\Theta(\sum_{i=1}^{p}(ki)) \le T_1(n) \le \Theta(\sum_{i=1}^{p+1}(ki))$.
And $\Theta(\sum_{i=1}^{p}(ki))) = \Theta(n^2), \Theta(\sum_{i=1}^{p+1}(ki))) = \Theta(n^2)$, which infers $T_1(n) = \Theta(n^2)$.

(b) (4') $T_2(n) = T_2(\alpha n) + T_2((1-\alpha)n) + \Theta(n)$

Solution:
W.L.O.G, assume $\alpha \le 0.5$.
Recurrence Tree Method (Suppose we recur till $n \le 1$):
The maximum depth of the tree is $\log_{\frac{1}{\alpha}} n = \Theta(\log n)$.
The minimum depth of the tree is $\log_{\frac{1}{1-\alpha}} n = \Theta(\log n)$.

For the nodes with the same depth in the recursion tree, recall that they sum up to n, so they cost $\Theta(n)$ totally without considering the recursion part. (Trivial proof of this is the use of induction for depths.)
Then we get that the total time $T_2(n)$ satisfies $\Theta(n \log_{\frac{1}{1-\alpha}} n) \leq T_2(n) \leq \Theta(n \log_{\frac{1}{\alpha}} n)$.
So $T_2(n) = \Theta(n \log n)$ since $\Theta(n \log_{\frac{1}{1-\alpha}} n) = \Theta(n \log n)$ and $\Theta(n \log_{\frac{1}{\alpha}} n) = \Theta(n \log n)$.

(c) (4') $T_3(n) = 2T_3(\sqrt{n}) + \Theta(\log n)$.

Solution: WLOG, we may assume that $n = 2^m$ for some $n \in \mathbb{Z}^+$
Construct a new function f such that $f(\log n) = T_3(n)$. Thus, we claim:

$$f(\log n) = T_3(n) = 2T_3(\sqrt{n}) + \Theta(\log n) = 2f(\frac{1}{2}\log n) + \Theta(\log n)$$

. Let $m = \log n$ and we get: $f(m) = 2f(\frac{m}{2}) + \Theta(m)$.
By the Master Theorem, we deduce that $f(m) = \Theta(m \log m)$.
Hence, we finally get: $T_3(n) = f(\log n) = \Theta(\log n \log \log n)$.

(d) (4') $T_4(n) = T_4(\alpha n) + T_4(\beta n) + \Theta(n), (0 < \alpha + \beta < 1, \alpha, \beta > 0)$
Hint:

1. Think out binomial theorem. You may try to expand the expression using the recurrence relation several times. Find out the similarities between them.
2. Cases in (b) are different from those in (d) since the geometric series $\sum_{i=0}^{\infty}(\alpha + \beta)^i$ converges only when $|\alpha + \beta| < 1$.

Solution: W.L.O.G, assume $\alpha \leq \beta$.
First, we show $T_4(n) = \sum_{i=0}^{k} \binom{k}{i} T_4(\alpha^i \beta^{k-i} n) + \sum_{i=0}^{k-1} \Theta((\alpha + \beta)^i n)$:
We use induction to prove it: (The induction hypothesis is the proposition above.)
When $k = 1$, the hypothesis holds. ($T_4(n) = T_4(\alpha n) + T_4(\beta n) + \Theta(n)$.)
Then expand the formula: $T_4(\alpha^i \beta^{k-i} n) = T_4(\alpha^{i+1} \beta^{k-i} n) + T_4(\alpha^i \beta^{k-i+1} n) + \Theta(\alpha^i \beta^{k-i} n)$.
That is:

- For $T_4(\alpha^{k+1} n)$ or $T_4(\beta^{k+1} n)$, the coefficients is 1, equal to $\binom{k+1}{0}(\binom{k+1}{k+1})$.
- For other items $T_4(\alpha^i \beta^{k+1-i} n)$, the coefficients came up with $T_4(\alpha^i \beta^{k-i} n)$ and $T_4(\alpha^{i-1} \beta^{k+1-i} n)$, equal to $\binom{k}{i} + \binom{k}{i-1} = \binom{k+1}{i}$.
- For extra constant item, we know $\Theta(\alpha^i \beta^j n)$ only appears when $T_4(\alpha^i \beta^j n)$ appears in last expansion. So the extra constant item is $\sum_{i=0}^{k-1} \binom{k-1}{i} \Theta(\alpha^i \beta^{k-1-i} n) = \Theta((\alpha + \beta)^{k-1} n)$.

Therefore, it holds for every k that

$$\begin{aligned} T_4(n) &= \sum_{i=0}^{k} \binom{k}{i} T_4(\alpha^i\beta^{k-i}n) + \sum_{i=0}^{k-2}\Theta((\alpha+\beta)^i n) + \Theta((\alpha+\beta)^{k-1}n) \\ &= \sum_{i=0}^{k} \binom{k}{i} T_4(\alpha^i\beta^{k-i}n) + \sum_{i=0}^{k-1}\Theta((\alpha+\beta)^i n) \end{aligned}$$

The last item $\sum_{i=0}^{k-1}\Theta((\alpha+\beta)^i n)$ converges to $\Theta(\frac{1}{1-\alpha-\beta}n) = \Theta(n)$ when $k \to +\infty$. And $k = 0$ it is $\Theta(n)$, so it is $\Theta(n)$ for every k due to its monotony increase of about k.

Denote the first item as C, i.e. $C(n) = \sum_{i=0}^{k}\binom{k}{i}T_4(\alpha^i\beta^{k-i}n)$, and $k_1 = \lceil -\log_\alpha n\rceil, k_2 = \lceil -\log_\beta n\rceil (k_1 \le k_2)$. Recall that for every k, $C(n)$ won't change. Then, on the one hand,

$$\begin{aligned} C(n) &= \sum_{i=0}^{k_1}\binom{k}{i}T_4(\alpha^i\beta^{k-i}n) \\ &\ge \sum_{i=0}^{k_1}\binom{k}{i}T_4(1) \\ &= \sum_{i=0}^{k_1}\binom{k}{i}1 \\ &= 2^{k_1} = 2^{\lceil -\log_\alpha n\rceil} \end{aligned}$$

On the other hand,

$$\begin{aligned} C(n) &= \sum_{i=0}^{k_2}\binom{k}{i}T_4(\alpha^i\beta^{k-i}n) \\ &\le \sum_{i=0}^{k_2}\binom{k}{i}T_4(1) \\ &= \sum_{i=0}^{k_2}\binom{k}{i}1 \\ &= 2^{k_2} = 2^{\lceil -\log_\beta n\rceil} \end{aligned}$$

Then $-\log_\alpha n \le \lceil -\log_\alpha n\rceil \le \log_2 C(n) \le \lceil -\log_\beta n\rceil \le -\log_\beta n + 1$, $\log_\alpha n = -\log_\alpha 2\log_2 n, -\log_\beta n + 1 = -\log_\beta 2\log_2 n + 1$, which are both $\Theta(\log_2 n)$, so that $\log_2 C(n) = \Theta(\log_2 n) \Rightarrow \exists c_1, c_2$ s.t. $c_1\log_2 n \le \log_2 C(n) \le c_2\log_2 n \Rightarrow 2^{c_1}n \le C(n) \le 2^{c_2}n$. Since c_1, c_2 are constant, we could obtain that $C(n) = \Theta(n)$.

Another estimation method for $C(n) = O(n)$:

We partially expand the formula s.t. $C(n) = \sum c_{i,j}T_4(\alpha^i\beta^j n)$, where (i, j) here must obey that either $\alpha^{i-1}\beta^j n$ or $\alpha^i\beta^{j-1}n$ is greater than 1 while $\alpha^i\beta^j n$ is no greater than 1. Note that we assume $\alpha \le \beta$, so the items are finite and $\alpha^{i-1}\beta^j n > 1$ for every n and its legal (i, j). Recall the expansion, we find that every expansion will abandon the total parameter size, so $\sum c_{i,j}\alpha^i\beta^j n \le n$, and $T_4(\alpha^i\beta^j n) \le 1$ as $\alpha^i\beta^j n \le 1$ with (i, j) legal, which shows that $C(n) \le O(n)$.

As we got $C(n) = \Theta(n)$ (or $O(n)$, which won't matter in the following contexts), we find out $T_4(n) = C(n) + \Theta(n) = \Theta(n)$.