IKEMKA ROMASON ROMANUS

BHU/20/04/05/0016

CMP 418 ASSIGNMENT

**Part A: Solving Recurrence**

**Problem 1-1**

**Group 0**

Arrange the following functions in increasing order of growth:

1. f1=n2f\_1 = n^2f1​=n2
2. f2=nf\_2 = nf2​=n
3. f3=nlog⁡nf\_3 = n \log nf3​=nlogn
4. f4=2nf\_4 = 2^nf4​=2n
5. f5=(log⁡n2)2f\_5 = (\log n^2)^2f5​=(logn2)2

**Solution:**

1. f2=nf\_2 = nf2​=n
2. f3=nlog⁡nf\_3 = n \log nf3​=nlogn
3. f5=(log⁡n2)2=4(log⁡n)2f\_5 = (\log n^2)^2 = 4 (\log n)^2f5​=(logn2)2=4(logn)2
4. f1=n2f\_1 = n^2f1​=n2
5. f4=2nf\_4 = 2^nf4​=2n

**Explanation:**

* nnn grows slower than nlog⁡nn \log nnlogn.
* (log⁡n2)2=4(log⁡n)2(\log n^2)^2 = 4 (\log n)^2(logn2)2=4(logn)2 grows slower than n2n^2n2 because log⁡n\log nlogn grows slower than nnn.
* 2n2^n2n grows exponentially, which is faster than any polynomial function like n2n^2n2.

**Group 1**

Arrange the following functions in increasing order of growth:

1. f1=log⁡((log⁡n)3)f\_1 = \log((\log n)^3)f1​=log((logn)3)
2. f2=(log⁡n)3log⁡(3n)f\_2 = (\log n)^3 \log(3n)f2​=(logn)3log(3n)
3. f3=3log⁡nf\_3 = 3 \log nf3​=3logn
4. f4=n3log⁡nf\_4 = n^3 \log nf4​=n3logn
5. f5=log⁡(3n)n3f\_5 = \log(3n) n^3f5​=log(3n)n3
6. f6=(log⁡log⁡n)3f\_6 = (\log \log n)^3f6​=(loglogn)3

**Solution:**

1. f6=(log⁡log⁡n)3f\_6 = (\log \log n)^3f6​=(loglogn)3
2. f1=log⁡((log⁡n)3)=3log⁡(log⁡n)f\_1 = \log((\log n)^3) = 3 \log (\log n)f1​=log((logn)3)=3log(logn)
3. f3=3log⁡nf\_3 = 3 \log nf3​=3logn
4. f2=(log⁡n)3log⁡(3n)f\_2 = (\log n)^3 \log(3n)f2​=(logn)3log(3n)
5. f4=n3log⁡nf\_4 = n^3 \log nf4​=n3logn
6. f5=log⁡(3n)n3f\_5 = \log(3n) n^3f5​=log(3n)n3

**Explanation:**

* (log⁡log⁡n)3(\log \log n)^3(loglogn)3 is the slowest growing function as double logarithms grow very slowly.
* log⁡((log⁡n)3)=3log⁡(log⁡n)\log((\log n)^3) = 3 \log (\log n)log((logn)3)=3log(logn) is slower than log⁡n\log nlogn.
* 3log⁡n3 \log n3logn is a simple logarithmic function.
* (log⁡n)3log⁡(3n)(\log n)^3 \log(3n)(logn)3log(3n) grows faster than log⁡n\log nlogn but slower than polynomial terms involving nnn.
* n3log⁡nn^3 \log nn3logn and log⁡(3n)n3\log(3n) n^3log(3n)n3 both have a cubic term but the latter has an additional logarithmic factor, making it the fastest growing in this group.

**Group 3**

Arrange the following functions in increasing order of growth:

1. f1=43nf\_1 = 43nf1​=43n
2. f2=2n4f\_2 = 2n^4f2​=2n4
3. f3=23n+1f\_3 = 23n + 1f3​=23n+1
4. f4=23nf\_4 = 23nf4​=23n
5. f5=25nf\_5 = 25nf5​=25n

**Solution:**

1. f1=43nf\_1 = 43nf1​=43n
2. f3=23n+1f\_3 = 23n + 1f3​=23n+1
3. f4=23nf\_4 = 23nf4​=23n
4. f5=25nf\_5 = 25nf5​=25n
5. f2=2n4f\_2 = 2n^4f2​=2n4

**Explanation:**

* 43n43n43n, 23n+123n + 123n+1, 23n23n23n, and 25n25n25n are all linear functions. They grow at the same rate asymptotically but with different coefficients.
* 2n42n^42n4 grows polynomially and faster than any linear function.

**Problem 1-2**

**(a) If t(n)∈O(g(n))t(n) \in O(g(n))t(n)∈O(g(n)), then g(n)∈Ω(t(n))g(n) \in \Omega(t(n))g(n)∈Ω(t(n)).**

**Solution:**

True. By definition, if t(n)∈O(g(n))t(n) \in O(g(n))t(n)∈O(g(n)), there exist positive constants ccc and n0n\_0n0​ such that for all n≥n0n \ge n\_0n≥n0​, t(n)≤c⋅g(n)t(n) \le c \cdot g(n)t(n)≤c⋅g(n). This implies g(n)g(n)g(n) is an upper bound on t(n)t(n)t(n), and hence, g(n)∈Ω(t(n))g(n) \in \Omega(t(n))g(n)∈Ω(t(n)), where Ω\OmegaΩ represents the lower bound.

**(b) Θ(αg(n))=Θ(g(n))\Theta(\alpha g(n)) = \Theta(g(n))Θ(αg(n))=Θ(g(n)), where α>0\alpha > 0α>0.**

**Solution:**

True. The constant multiplier α\alphaα does not change the asymptotic behavior of g(n)g(n)g(n). Therefore, Θ(αg(n))=Θ(g(n))\Theta(\alpha g(n)) = \Theta(g(n))Θ(αg(n))=Θ(g(n)).

**(c) Θ(g(n))=O(g(n))∩Ω(g(n))\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))Θ(g(n))=O(g(n))∩Ω(g(n)).**

**Solution:**

True. By definition, Θ(g(n))\Theta(g(n))Θ(g(n)) represents functions that are both upper and lower bounded by g(n)g(n)g(n) within constant factors. Thus, Θ(g(n))\Theta(g(n))Θ(g(n)) is the intersection of O(g(n))O(g(n))O(g(n)) and Ω(g(n))\Omega(g(n))Ω(g(n)).

**(d) For any two nonnegative functions t(n)t(n)t(n) and g(n)g(n)g(n) defined on the set of nonnegative integers, either t(n)∈O(g(n))t(n) \in O(g(n))t(n)∈O(g(n)), or t(n)∈Ω(g(n))t(n) \in \Omega(g(n))t(n)∈Ω(g(n)), or both.**

**Solution:**

False. There are cases where t(n)t(n)t(n) and g(n)g(n)g(n) do not asymptotically bound each other, especially if they oscillate or grow differently for different subsets of nnn. For example, if t(n)t(n)t(n) is a linear function for even nnn and quadratic for odd nnn, and g(n)g(n)g(n) is quadratic for even nnn and linear for odd nnn, then neither function asymptotically bounds the other.

**Problem 2-1: Back-solving recurrences**

**(a) If T(n)=Θ(n2log⁡n)T(n) = \Theta(n^2 \log n)T(n)=Θ(n2logn) is a solution to T(n)=aT(n/2)+Θ(n2)T(n) = aT(n/2) + \Theta(n^2)T(n)=aT(n/2)+Θ(n2), where aaa is a positive integer, find all possible values of aaa.**

**Solution:**

Given the recurrence T(n)=aT(n/2)+Θ(n2)T(n) = aT(n/2) + \Theta(n^2)T(n)=aT(n/2)+Θ(n2):

1. Use the Master Theorem for divide-and-conquer recurrences of the form T(n)=aT(n/b)+f(n)T(n) = aT(n/b) + f(n)T(n)=aT(n/b)+f(n).
2. Identify the values: a=aa = aa=a, b=2b = 2b=2, and f(n)=Θ(n2)f(n) = \Theta(n^2)f(n)=Θ(n2).

According to the Master Theorem:

* If f(n)=O(nc)f(n) = O(n^c)f(n)=O(nc) where c<log⁡bac < \log\_b ac<logb​a, then T(n)=Θ(nlog⁡ba)T(n) = \Theta(n^{\log\_b a})T(n)=Θ(nlogb​a).
* If f(n)=Θ(nc)f(n) = \Theta(n^c)f(n)=Θ(nc) where c=log⁡bac = \log\_b ac=logb​a, then T(n)=Θ(nclog⁡n)T(n) = \Theta(n^c \log n)T(n)=Θ(nclogn).
* If f(n)=Ω(nc)f(n) = \Omega(n^c)f(n)=Ω(nc) where c>log⁡bac > \log\_b ac>logb​a, and af(n/b)≤kf(n)a f(n/b) \le k f(n)af(n/b)≤kf(n) for some k<1k < 1k<1, then T(n)=Θ(f(n))T(n) = \Theta(f(n))T(n)=Θ(f(n)).

Here, f(n)=Θ(n2)f(n) = \Theta(n^2)f(n)=Θ(n2) and T(n)=Θ(n2log⁡n)T(n) = \Theta(n^2 \log n)T(n)=Θ(n2logn), so:

* n2log⁡n=nlog⁡2alog⁡nn^2 \log n = n^{\log\_2 a} \log nn2logn=nlog2​alogn

Thus, log⁡2a=2\log\_2 a = 2log2​a=2 implies a=22=4a = 2^2 = 4a=22=4.

Therefore, the value of aaa is **4**.

**(b) If T(n)=Θ(n2)T(n) = \Theta(n^2)T(n)=Θ(n2) is a solution to T(n)=aT(n/3)+Θ(n)T(n) = aT(n/3) + \Theta(n)T(n)=aT(n/3)+Θ(n), where aaa is a positive integer, find all possible values of aaa.**

**Solution:**

Given the recurrence T(n)=aT(n/3)+Θ(n)T(n) = aT(n/3) + \Theta(n)T(n)=aT(n/3)+Θ(n):

1. Use the Master Theorem for divide-and-conquer recurrences of the form T(n)=aT(n/b)+f(n)T(n) = aT(n/b) + f(n)T(n)=aT(n/b)+f(n).
2. Identify the values: a=aa = aa=a, b=3b = 3b=3, and f(n)=Θ(n)f(n) = \Theta(n)f(n)=Θ(n).

According to the Master Theorem:

* If f(n)=O(nc)f(n) = O(n^c)f(n)=O(nc) where c<log⁡bac < \log\_b ac<logb​a, then T(n)=Θ(nlog⁡ba)T(n) = \Theta(n^{\log\_b a})T(n)=Θ(nlogb​a).
* If f(n)=Θ(nc)f(n) = \Theta(n^c)f(n)=Θ(nc) where c=log⁡bac = \log\_b ac=logb​a, then T(n)=Θ(nclog⁡n)T(n) = \Theta(n^c \log n)T(n)=Θ(nclogn).
* If f(n)=Ω(nc)f(n) = \Omega(n^c)f(n)=Ω(nc) where c>log⁡bac > \log\_b ac>logb​a, and af(n/b)≤kf(n)a f(n/b) \le k f(n)af(n/b)≤kf(n) for some k<1k < 1k<1, then T(n)=Θ(f(n))T(n) = \Theta(f(n))T(n)=Θ(f(n)).

Here, f(n)=Θ(n)f(n) = \Theta(n)f(n)=Θ(n) and T(n)=Θ(n2)T(n) = \Theta(n^2)T(n)=Θ(n2), so:

* n2=nlog⁡3an^2 = n^{\log\_3 a}n2=nlog3​a

Thus, log⁡3a=2\log\_3 a = 2log3​a=2 implies a=32=9a = 3^2 = 9a=32=9.

Therefore, the value of aaa is **9**.

**(c) If T(n)=Θ(n2)T(n) = \Theta(n^2)T(n)=Θ(n2) is a solution to T(n)=4T(n/b)+Θ(n2)T(n) = 4T(n/b) + \Theta(n^2)T(n)=4T(n/b)+Θ(n2), where bbb is a positive real number, find all possible values of bbb.**

**Solution:**

Given the recurrence T(n)=4T(n/b)+Θ(n2)T(n) = 4T(n/b) + \Theta(n^2)T(n)=4T(n/b)+Θ(n2):

1. Use the Master Theorem for divide-and-conquer recurrences of the form T(n)=aT(n/b)+f(n)T(n) = aT(n/b) + f(n)T(n)=aT(n/b)+f(n).
2. Identify the values: a=4a = 4a=4, b=bb = bb=b, and f(n)=Θ(n2)f(n) = \Theta(n^2)f(n)=Θ(n2).

According to the Master Theorem:

* If f(n)=O(nc)f(n) = O(n^c)f(n)=O(nc) where c<log⁡bac < \log\_b ac<logb​a, then T(n)=Θ(nlog⁡ba)T(n) = \Theta(n^{\log\_b a})T(n)=Θ(nlogb​a).
* If f(n)=Θ(nc)f(n) = \Theta(n^c)f(n)=Θ(nc) where c=log⁡bac = \log\_b ac=logb​a, then T(n)=Θ(nclog⁡n)T(n) = \Theta(n^c \log n)T(n)=Θ(nclogn).
* If f(n)=Ω(nc)f(n) = \Omega(n^c)f(n)=Ω(nc) where c>log⁡bac > \log\_b ac>logb​a, and af(n/b)≤kf(n)a f(n/b) \le k f(n)af(n/b)≤kf(n) for some k<1k < 1k<1, then T(n)=Θ(f(n))T(n) = \Theta(f(n))T(n)=Θ(f(n)).

Here, f(n)=Θ(n2)f(n) = \Theta(n^2)f(n)=Θ(n2) and T(n)=Θ(n2)T(n) = \Theta(n^2)T(n)=Θ(n2), so:

* n2=nlog⁡b4n^2 = n^{\log\_b 4}n2=nlogb​4

Thus, log⁡b4=2\log\_b 4 = 2logb​4=2 implies 4=b24 = b^24=b2 or b=4=2b = \sqrt{4} = 2b=4​=2.

Therefore, the value of bbb is **2**.

**(d) If T(n)=Θ(n6.006)T(n) = \Theta(n^{6.006})T(n)=Θ(n6.006) is a solution to T(n)=5T(n/b)+Θ(n5)T(n) = 5T(n/b) + \Theta(n^5)T(n)=5T(n/b)+Θ(n5), where bbb is a positive real number, find all possible values of bbb.**

**Solution:**

Given the recurrence T(n)=5T(n/b)+Θ(n5)T(n) = 5T(n/b) + \Theta(n^5)T(n)=5T(n/b)+Θ(n5):

1. Use the Master Theorem for divide-and-conquer recurrences of the form T(n)=aT(n/b)+f(n)T(n) = aT(n/b) + f(n)T(n)=aT(n/b)+f(n).
2. Identify the values: a=5a = 5a=5, b=bb = bb=b, and f(n)=Θ(n5)f(n) = \Theta(n^5)f(n)=Θ(n5).

According to the Master Theorem:

* If f(n)=O(nc)f(n) = O(n^c)f(n)=O(nc) where c<log⁡bac < \log\_b ac<logb​a, then T(n)=Θ(nlog⁡ba)T(n) = \Theta(n^{\log\_b a})T(n)=Θ(nlogb​a).
* If f(n)=Θ(nc)f(n) = \Theta(n^c)f(n)=Θ(nc) where c=log⁡bac = \log\_b ac=logb​a, then T(n)=Θ(nclog⁡n)T(n) = \Theta(n^c \log n)T(n)=Θ(nclogn).
* If f(n)=Ω(nc)f(n) = \Omega(n^c)f(n)=Ω(nc) where c>log⁡bac > \log\_b ac>logb​a, and af(n/b)≤kf(n)a f(n/b) \le k f(n)af(n/b)≤kf(n) for some k<1k < 1k<1, then T(n)=Θ(f(n))T(n) = \Theta(f(n))T(n)=Θ(f(n)).

Here, f(n)=Θ(n5)f(n) = \Theta(n^5)f(n)=Θ(n5) and T(n)=Θ(n6.006)T(n) = \Theta(n^{6.006})T(n)=Θ(n6.006), so:

* n6.006=nlog⁡b5n^{6.006} = n^{\log\_b 5}n6.006=nlogb​5

Thus, log⁡b5=6.006\log\_b 5 = 6.006logb​5=6.006 implies b=51/6.006b = 5^{1/6.006}b=51/6.006.

Therefore, the value of bbb is **51/6.0065^{1/6.006}51/6.006**.

**(e) If T(n)=Θ(n2)T(n) = \Theta(n^2)T(n)=Θ(n2) is a solution to T(n)=6T(n/6)+f(n)T(n) = 6T(n/6) + f(n)T(n)=6T(n/6)+f(n), find one possible function fff.**

**Solution:**

Given the recurrence T(n)=6T(n/6)+f(n)T(n) = 6T(n/6) + f(n)T(n)=6T(n/6)+f(n):

1. Use the Master Theorem for divide-and-conquer recurrences of the form T(n)=aT(n/b)+f(n)T(n) = aT(n/b) + f(n)T(n)=aT(n/b)+f(n).
2. Identify the values: a=6a = 6a=6, b=6b = 6b=6, and we need to find f(n)f(n)f(n) such that T(n)=Θ(n2)T(n) = \Theta(n^2)T(n)=Θ(n2).

According to the Master Theorem:

* If f(n)=O(nc)f(n) = O(n^c)f(n)=O(nc) where c<log⁡bac < \log\_b ac<logb​a, then T(n)=Θ(nlog⁡ba)T(n) = \Theta(n^{\log\_b a})T(n)=Θ(nlogb​a).
* If f(n)=Θ(nc)f(n) = \Theta(n^c)f(n)=Θ(nc) where c=log⁡bac = \log\_b ac=logb​a, then T(n)=Θ(nclog⁡n)T(n) = \Theta(n^c \log n)T(n)=Θ(nclogn).
* If f(n)=Ω(nc)f(n) = \Omega(n^c)f(n)=Ω(nc) where c>log⁡bac > \log\_b ac>logb​a, and af(n/b)≤kf(n)a f(n/b) \le k f(n)af(n/b)≤kf(n) for some k<1k < 1k<1, then T(n)=Θ(f(n))T(n) = \Theta(f(n))T(n)=Θ(f(n)).

Here, T(n)=Θ(n2)T(n) = \Theta(n^2)T(n)=Θ(n2) and log⁡66=1\log\_6 6 = 1log6​6=1. So we need f(n)f(n)f(n) such that:

* n2=f(n)n^2 = f(n)n2=f(n)

Thus, f(n)=Θ(n2)f(n) = \Theta(n^2)f(n)=Θ(n2).

Therefore, one possible function f(n)f(n)f(n) is Θ(n2)\Theta(n^2)Θ(n2).

**Part B: Sorting Problem**

**Problem 2-1**

Use the given algorithm ComparisonCountingSort to sort the array [50,30,80,10,40,60,20][50, 30, 80, 10, 40, 60, 20][50,30,80,10,40,60,20].

**Solution:**

1. Initialize Count array to zeros.
2. Perform comparisons to count the position for each element.
3. Place elements in the S array based on counts.

**Steps:**

Initial array: [50,30,80,10,40,60,20][50, 30, 80, 10, 40, 60, 20][50,30,80,10,40,60,20]

Count array after comparisons:

* Count[0]=6\text{Count}[0] = 6Count[0]=6 (50 is greater than 6 elements)
* Count[1]=3\text{Count}[1] = 3Count[1]=3 (30 is greater than 3 elements)
* Count[2]=6\text{Count}[2] = 6Count[2]=6 (80 is greater than 6 elements)
* Count[3]=0\text{Count}[3] = 0Count[3]=0 (10 is greater than 0 elements)
* Count[4]=4\text{Count}[4] = 4Count[4]=4 (40 is greater than 4 elements)
* Count[5]=5\text{Count}[5] = 5Count[5]=5 (60 is greater than 5 elements)
* Count[6]=1\text{Count}[6] = 1Count[6]=1 (20 is greater than 1 element)

Array S after sorting based on counts: S=[10,20,30,40,50,60,80]S = [10, 20, 30, 40, 50, 60, 80]S=[10,20,30,40,50,60,80]

**Basic Operation:**

Comparisons between elements.

**Computing:**

The algorithm counts the number of elements less than each element to determine their sorted position.

**Stability:**

No, the algorithm is not stable as it does not maintain the relative order of equal elements.

**In-place:**

No, the algorithm uses extra space for the Count and S arrays.

**Complexity:**

The time complexity is O(n2)O(n^2)O(n2) due to the nested loops for counting comparisons.

**Problem 2-2: Quicksort Example**

**Array**: [E, X, A, M, P, L, E, S]

* Choose pivot (typically the last element, S).
* Partition the array around the pivot.
* Recursively apply quicksort to the partitions.

**Part C: Searching Problem**

**Problem 3-1: Brute Force Algorithm in Binary Tree**

**(a) Comparisons for pattern 00001**:

* The algorithm will check each substring of length 5.
* For 1000 zeros, there are 1000−5+1=9961000 - 5 + 1 = 9961000−5+1=996 comparisons.

**(b) Comparisons for pattern 01010**:

* Similarly, there will be 996 comparisons.

**Brute Force Analysis**

* **Strengths**: Simplicity, easy to implement, works well for small inputs.
* **Weaknesses**: Inefficient for large inputs, high time complexity.

**Minimum Comparisons for BRANDING**:

* Text: THERE\_IS\_MORE\_TO\_LIFE\_THAN\_INCREASING\_ITS\_SPEED
* Minimum comparisons: Length of text −-− length of pattern +1+ 1+1.

### Problem 3-2

**Given List**: 1, 8, 6, 5, 3, 7, 4, 2

#### Part (i): Construct a Heap by Successive Key Insertions

To build a max-heap by successive insertions, we insert each key and ensure the heap property (parent node is larger than child nodes) is maintained.

1. Insert 1:

Copy code

1

1. Insert 8:

Copy code

8

/

1

1. Insert 6:

Copy code

8

/ \

1 6

1. Insert 5:

Copy code

8

/ \

1 6

/ 5

markdown

Copy code

5. Insert 3:

Copy code

8

/ \

1 6 /  
5 3

markdown

Copy code

6. Insert 7:

Copy code

8

/ \

1 7 / \ / 5 3 6

markdown

Copy code

7. Insert 4:

Copy code

8

/ \

1 7 / \ /  
5 3 6 4

markdown

Copy code

8. Insert 2:

Copy code

8

/ \

1 7 / \ /  
5 3 6 4 / 2

makefile

Copy code

\*\*Max-Heap\*\*:

Copy code

8

/ \

5 7 / \ /  
1 3 6 4 / 2

bash

Copy code

#### Part (ii): Heapsort

To sort the list using heapsort, we repeatedly extract the maximum element and rebuild the heap.

1. Max-Heap:

Copy code

8

/ \

5 7 / \ /  
1 3 6 4 / 2

java

Copy code

2. Swap root (8) with the last element (2) and rebuild the heap (size reduced by 1):

Copy code

7

/ \

5 6 / \ /  
1 3 2 4

sql

Copy code

3. Swap root (7) with the last element (4) and rebuild the heap:

Copy code

6

/ \

5 4 / \ /  
1 3 2 7

sql

Copy code

4. Swap root (6) with the last element (2) and rebuild the heap:

Copy code

5

/ \

3 4 / \ / 1 2 6 7

sql

Copy code

5. Swap root (5) with the last element (1) and rebuild the heap:

Copy code

4

/ \

3 2 / \  
1 5 6 7

sql

Copy code

6. Swap root (4) with the last element (1) and rebuild the heap:

Copy code

3

/ \

1 2 / 4 5 6 7

sql

Copy code

7. Swap root (3) with the last element (2) and rebuild the heap:

Copy code

2

/ \

1 3 / 4 5 6 7

sql

Copy code

8. Swap root (2) with the last element (1) and rebuild the heap:

Copy code

1

/

2 3 / 4 5 6 7

yaml

Copy code

\*\*Sorted List\*\*: 1, 2, 3, 4, 5, 6, 7, 8

---

### Problem 3-3

\*\*Initial Array\*\*: 68, 55, 44, 79, 19, 9

#### Algorithm Trace:

Initial Array: 68 55 44 79 19 9

Pass 1: 68 55 44 79 19 9 -> no swap needed

Pass 2: 68 55 44 79 19 9 -> no swap needed

Pass 3: 68 55 79 44 19 9 -> swap 79 and 44

Pass 4: 68 79 55 44 19 9 -> swap 79 and 55 79 68 55 44 19 9 -> swap 79 and 68

Pass 5: 79 68 55 44 19 9 -> no swap needed

Pass 6: 79 68 55 44 19 9 -> no swap needed

\*\*(a) Basic Operations\*\*:

The basic operations are comparisons and swaps of elements.

\*\*(b) Complexity Class\*\*:

The algorithm performs in \(O(n^2)\) time complexity due to the nested loops.

\*\*(c) Is the Algorithm In-Place?\*\*:

Yes, the algorithm is in-place as it only swaps elements within the array without using extra space.

\*\*(d) Is the Algorithm Stable?\*\*:

No, the algorithm is not stable. To make it stable, we need to modify the condition in the while loop to ensure that elements with equal values maintain their relative order. Change `A[j] ≤ A[j+1]` to `A[j] < A[j+1]`.

---

These solutions should comprehensively address Problems 3-2 and 3-3 from the document. If you need further details or additional problems solved, feel free to ask!

**Problem 4-1: Knapsack Problem**

Given nnn items with known weights w1,w2,…,wnw\_1, w\_2, \ldots, w\_nw1​,w2​,…,wn​ and values v1,v2,…,vnv\_1, v\_2, \ldots, v\_nv1​,v2​,…,vn​, and a knapsack of capacity WWW, find the most valuable subset of the items that fit into the knapsack.

**Items and their respective weights and values:**

| **Item No.** | **Item Weight** | **Item Value** |
| --- | --- | --- |
| 1 | 2 | $15 |
| 2 | 3 | $20 |
| 3 | 1 | $10 |
| 4 | 2 | $12 |

Assume a knapsack of capacity 5 units of weight.

**(a) Exhaustive Search Solution:**

Exhaustive search involves evaluating all possible subsets of items to find the one with the maximum value that fits within the knapsack's capacity.

Possible subsets and their total weights and values:

1. {Item 1}: Weight = 2, Value = 15
2. {Item 2}: Weight = 3, Value = 20
3. {Item 3}: Weight = 1, Value = 10
4. {Item 4}: Weight = 2, Value = 12
5. {Item 1, Item 2}: Weight = 5, Value = 35
6. {Item 1, Item 3}: Weight = 3, Value = 25
7. {Item 1, Item 4}: Weight = 4, Value = 27
8. {Item 2, Item 3}: Weight = 4, Value = 30
9. {Item 2, Item 4}: Weight = 5, Value = 32
10. {Item 3, Item 4}: Weight = 3, Value = 22
11. {Item 1, Item 2, Item 3}: Weight = 6 (exceeds capacity)
12. {Item 1, Item 2, Item 4}: Weight = 7 (exceeds capacity)
13. {Item 1, Item 3, Item 4}: Weight = 5, Value = 37 (maximum value)
14. {Item 2, Item 3, Item 4}: Weight = 6 (exceeds capacity)

The most valuable subset of the items that fits into the knapsack using exhaustive search is {Item 1, Item 3, Item 4} with a total weight of 5 and a total value of 37.

**(b) Dynamic Programming Solution:**

Define V[i,w]V[i, w]V[i,w] as the maximum value that can be obtained with the first iii items and a knapsack capacity www. The recurrence relation is: V[i,w]=max⁡(V[i−1,w],V[i−1,w−wi]+vi)V[i, w] = \max(V[i-1, w], V[i-1, w-w\_i] + v\_i)V[i,w]=max(V[i−1,w],V[i−1,w−wi​]+vi​)

Here's the step-by-step solution using dynamic programming:

Initialize the DP table VVV with dimensions (number of items + 1) x (capacity + 1): V[0,w]=0 for all wV[0, w] = 0 \text{ for all } wV[0,w]=0 for all w V[i,0]=0 for all iV[i, 0] = 0 \text{ for all } iV[i,0]=0 for all i

| **i\w** | **0** | **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 15 | 15 | 15 | 15 |
| 2 | 0 | 0 | 15 | 20 | 20 | 35 |
| 3 | 0 | 10 | 15 | 25 | 30 | 35 |
| 4 | 0 | 10 | 15 | 25 | 27 | 37 |

The optimal value for capacity 5 is 37, achieved with the subset {Item 1, Item 3, Item 4}.

**Problem 4-2: Longest Common Subsequence (LCS)**

Given two sequences, find the longest subsequence present in both.

**Example:**

For strings "KADUNA" and "KANO":

* LCS(KADUNA, KANO) = KAN

**(a) Recurrence Relation:**

Let XXX and YYY be two sequences, and L[i,j]L[i, j]L[i,j] be the length of LCS of X[1…i]X[1 \ldots i]X[1…i] and Y[1…j]Y[1 \ldots j]Y[1…j]. L[i,j]={0if i=0 or j=0L[i−1,j−1]+1if X[i]=Y[j]max⁡(L[i−1,j],L[i,j−1])if X[i]≠Y[j]L[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ L[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \\ \max(L[i-1, j], L[i, j-1]) & \text{if } X[i] \neq Y[j] \end{cases}L[i,j]=⎩⎨⎧​0L[i−1,j−1]+1max(L[i−1,j],L[i,j−1])​if i=0 or j=0if X[i]=Y[j]if X[i]=Y[j]​

**(b) Algorithm to Compute LCS Length:**

def LCS\_length(X, Y):

m = len(X)

k = len(Y)

L = [[0] \* (k + 1) for i in range(m + 1)]

for i in range(m + 1):

for j in range(k + 1):

if i == 0 or j == 0:

L[i][j] = 0

elif X[i - 1] == Y[j - 1]:

L[i][j] = L[i - 1][j - 1] + 1

else:

L[i][j] = max(L[i - 1][j], L[i][j - 1])

return L[m][k]

**(c) Example with "KADUNA" and "KAKNO":**

Fill the DP table for strings "KADUNA" and "KAKNO":

|  |  | **K** | **A** | **K** | **N** | **O** |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 0 | 0 | 0 | 0 | 0 |
| K | 0 | 1 | 1 | 1 | 1 | 1 |
| A | 0 | 1 | 2 | 2 | 2 | 2 |
| D | 0 | 1 | 2 | 2 | 2 | 2 |
| U | 0 | 1 | 2 | 2 | 2 | 2 |
| N | 0 | 1 | 2 | 2 | 3 | 3 |
| A | 0 | 1 | 2 | 2 | 3 | 3 |

The length of the LCS is 3, which corresponds to "KAN".

**Problem 4-3: Sorting Books and Removing Duplicates**

**(a) Proposed Algorithm:**

def sort\_and\_remove\_duplicates(books):

books.sort() # Step 1: Sort the books

unique\_books = [] # Step 2: Initialize list for unique books

for book in books:

if not unique\_books or unique\_books[-1] != book:

unique\_books.append(book) # Step 3: Add if not duplicate

return unique\_books

**(b) Is the proposed algorithm in place?**

No, the algorithm uses additional space for the unique\_books list, so it is not in-place.

**(c) Performance Evaluation:**

Sorting the books takes O(nlog⁡n)O(n \log n)O(nlogn) time, and the pass to remove duplicates takes O(n)O(n)O(n) time. The overall complexity is O(nlog⁡n)O(n \log n)O(nlogn).

**(d) Alternative Solution:**

Sort the books and then remove duplicates by checking consecutive books.

* Complexity of sorting: O(nlog⁡n)O(n \log n)O(nlogn)
* Complexity of checking consecutive books: O(n)O(n)O(n)
* Overall complexity: O(nlog⁡n)O(n \log n)O(nlogn)

**Problem 4-4: Graph Traversal**

Given the following graph, perform the traversals.

**(b) Depth-first Traversal from vertex 'a':**

Order: a, b, d, e, c, f

**(c) Breadth-first Traversal from vertex 'a':**

Order: a, b, c, d, e, f

**(d) Topological Ordering using Source Removal Algorithm:**

One possible topological order: a, b, d, c, e, f