1) განვიხილოთ სწრაფი სორტიება (quick sort) და ყალბი სორტირება(bubble sort). თითოეული ალგორითმისთვის რა იქნება ყველაზე ცუდი შემთხვევის ასიმპრომური ზედა ზღვარი თუ ვიცით რომ

ა) შესატანი მონაცემები წინასწარ სორტირებულია

ბ) შესატანი მონაცემები შებრუნებულად არის სორტირებული

ც) შესატანი მონაცემები შეიცავს ერთიდაიგივე რიცხვის "ი"-ცალ ასლს

პასუხი:

ა) როგორც სწრაფი დალაგების, ასევე ბუშტების დალაგებისთვის, ყველაზე უარესი ასიმპტოტური ზღვარი, თუ შეყვანის მონაცემები წინასწარ არის დახარისხებული, არის O(n^2). სწრაფი დალაგებისას, ეს ხდება მაშინ, როდესაც საყრდენი ელემენტი თანმიმდევრულად არის არჩეული, როგორც ყველაზე პატარა ან ყველაზე დიდი ელემენტი დანაყოფში, რაც იწვევს გაუწონასწორებელ დანაყოფებს. ბუშტების დალაგებისას, წინასწარ დახარისხებული მონაცემები განაპირობებს ალგორითმს შეასრულოს მაქსიმალური რაოდენობის შედარებები და გაცვლა, რაც გამოიწვევს დროის კვადრატულ სირთულეს.

ბ) სწრაფი დალაგებისთვის, უარეს შემთხვევაში ასიმპტომური ზედა ზღვარი, თუ შეყვანის მონაცემები საპირისპიროა დალაგებული, ასევე არის O(n^2), იგივე მიზეზით, როგორც წინასწარ დახარისხებულ შემთხვევაში. თუმცა, ბუშტების დალაგებისთვის, უარეს შემთხვევაში ასიმპტოტური ზედა ზღვარი, თუ შეყვანის მონაცემები საპირისპიროა დალაგებული, კვლავ არის O(n^2), მაგრამ სვოპების რაოდენობა ნაკლები იქნება, ვიდრე წინასწარ დალაგებულ შემთხვევაში.

გ) სწრაფი დალაგებისთვის, უარეს შემთხვევაში ასიმპტოტური ზედა ზღვარი, თუ შესაყვანი მონაცემები შეიცავს იგივე რიცხვის "i" ასლს არის O(n^2), თუ კრებსითი შერჩევის სტრატეგია არ არის შეცვლილი. ამ შემთხვევაში, pivot ყოველთვის იქნება პირველი ან ბოლო ელემენტი დანაყოფში, რაც იწვევს გაუწონასწორებელ დანაყოფებს. ბუშტების დალაგებისთვის, უარეს შემთხვევაში ასიმპტოტური ზედა ზღვარი, თუ შესასვლელი მონაცემი შეიცავს იგივე რიცხვის "i" ასლს არის O(n^2), რადგან ალგორითმი შეასრულებს შედარებებისა და სვოპების მაქსიმალურ რაოდენობას.

2)

მოცემულია ნ ზომის მონაცემთა მასივი და მთელი რიცხვი კ. დაწერეთ პროგრამა რომელიც დაალაგებს მასივს და ორობითი ძიების გამოყენებით იპოვის კ რიცხვის პოზიციას მასივში. დაამატეთ იტერაციათა მთვლელი და შეაფასეთ თუ რამდენ იტერაციაში გადაიჭრება პრობლემა (ფუნქციებიც თავად შეადგინეთ წინააღმდეგ შემთხვევაში საკითხი შესრულებულად არ ჩაითვლება)

import random

def quick\_sort(arr):

if len(arr) <= 1:

return arr

pivot = arr[random.randint(0, len(arr) - 1)]

left = [x for x in arr if x < pivot]

middle = [x for x in arr if x == pivot]

right = [x for x in arr if x > pivot]

return quick\_sort(left) + middle + quick\_sort(right)

def binary\_search(arr, K):

low = 0

high = len(arr) - 1

iterations = 0

while low <= high:

iterations += 1

mid = (low + high) // 2

if arr[mid] == K:

return mid, iterations

elif arr[mid] < K:

low = mid + 1

else:

high = mid - 1

return -1, iterations

def estimate\_iterations(N, K, data\_type):

arr = [i for i in range(N)]

if data\_type == "pre-sorted":

pass

elif data\_type == "reverse-sorted":

arr = arr[::-1]

elif data\_type == "i-copies":

for i in range(N // 2):

arr[i] = K

else:

raise ValueError("Invalid data type")

sorted\_arr = quick\_sort(arr)

pos, iterations = binary\_search(sorted\_arr, K)

if pos == -1:

print(f"{K} is not in the array")

else:

print(f"{K} is at position {pos} in the sorted array")

print(f"Estimated number of iterations: {iterations}")

N = 10

K = 5

data\_type = "pre-sorted"

estimate\_iterations(N, K, data\_type)

3)

რა არის ბმულის სიების უპირატესობა სხვა სახის მონაცემთა სტრუქტურებთან შედარებით?

ა მეხსიერების დაზოგვა და მარტივი წვდომა

4)

აღწერეთ გათიშე და იბატონე ალგორითმი შეადარე ის დინამიურ პროგრამირებასთან და აღწერე განსხვავება. მოიყვანე მაგალითები

პასუხი:

გადართვა და იბატონე ალგორითმი არის გაყოფა და იბატონე ალგორითმის განზოგადება, რომელსაც შეუძლია გაუმკლავდეს სიტუაციებს, როდესაც პრობლემის მრავალი გადაწყვეტა არსებობს. გადართვის და დაპყრობის ალგორითმში პრობლემა იყოფა მრავალ ქვეპრობლემად და თითოეული ქვეპრობლემა დამოუკიდებლად წყდება. შემდეგ გადაწყვეტილებები გაერთიანებულია ორიგინალური პრობლემის გადაჭრის შესაქმნელად.

გაყავი და იბატონე ალგორითმის მსგავსად, გადართვა და დაამარცხე ალგორითმი ასევე გამოიყენება დამოუკიდებელი ქვეპრობლემების პრობლემებისთვის. ამ ორ მიდგომას შორის მთავარი განსხვავება ისაა, რომ გადართვის და დაპყრობის ალგორითმებს შეუძლიათ პრობლემების გადაჭრა მრავალი გადაწყვეტით, ხოლო გაყოფა და დაიპყრო ალგორითმები, როგორც წესი, აწარმოებენ მხოლოდ ერთ ამოხსნას.

გადართვის და დაპყრობის ალგორითმის მაგალითია ჰანოის კოშკი. ამ პრობლემაში მიზანია დისკების ნაკრების გადატანა ერთი კოშკიდან მეორეში, კონკრეტული წესების დაცვით. პრობლემა შეიძლება დაიყოს რამდენიმე ქვეპრობლემად, თითოეული წარმოადგენს ერთი დისკის მოძრაობას. ამ ქვეპრობლემების გადაწყვეტილებები შემდეგ გაერთიანებულია თავდაპირველი პრობლემის გადაჭრის შესაქმნელად.

5)

მოცემულია სიმრავლე რომელიც შეიცავს როგორც დადებით ასევე უარყოფით და ნულოვან ელემენტებს იპოვეთ მაქსიმალური ნამრავლის მომცემი ქვესიმრავლე arr = [-2,-4,-5,0,60]

def max\_product\_subset(nums):

if not nums:

return 0

max\_product = max(nums[0], 0)

current\_product = nums[0]

for num in nums[1:]:

if num == 0:

current\_product = 0

elif num > 0:

current\_product = current\_product \* num

else:

current\_product = max(current\_product \* num, num)

max\_product = max(max\_product, current\_product)

return max\_product

# Example usage:

arr = [-2,-4,-5,0,60]

print(max\_product\_subset(arr))