@author: Shada Al-Wakkal, Git:Shada12354

Sammanfattning

Färgkod: variabel, funktion, loop & modul.
Denna session demonstrerar koncept som sorteringsalgoritmer, rekursion, implementering av matematiska sekvenser och förståelse av programmeringskoncept.

Kod

13.3.2

Detta avsnitt introducerar och implementerar merge sort-algoritmen, en effektiv sorteringsmetod.

MERGE()

- Sammanfogar, “mergar”, två sorterade listor lst1 och lst2 till en tredje lista lst3.
- i1, i2, och i3 är indexvariabler för att hålla koll på positionerna i de tre listorna.
- while-loop används för iterera genom lst1 och lst2 och placerar de mindre elementen i lst3 i sorterad ordning.

MERGESORT()

- Delar upp listan i två halvor (nums1 och nums2).
- Anropar rekursivt mergeSort på nums1 och nums2.
- Använder merge-funktionen för att slå samman de sorterade halvorna tillbaka i nums.

13.4.1

Detta avsnitt löser det klassiska "Tower of Hanoi"-problemet med hjälp av rekursion.

MOVETOWER()

- Flyttar en torn av **n** brickor från source till dest, med hjälp av temp.
- Basfall: Om **n** är 1, flyttas en bricka direkt från source till dest.
- Rekursivt steg: Flyttar **n-1** brickor från source till temp, sedan den sista brickan från source till dest, och till sist **n-1** brickor från temp till dest.

HANOI()

- Enkelt anrop för att lösa "Tower of Hanoi"-problemet med **n** brickor. Använder moveTower-funktionen.

13.6 - PROGRAMMING EXERCISES 1)

FIB()

- Beräknar det **n-te** Fibonacci-talet.
- Basfall: Om **n** är 0 eller 1, returneras **n** direkt.
- Rekursivt steg: Beräknar det **n-1** och **n-2** Fibonacci-talet genom rekursiva anrop och returnerar deras summa.

13.6 - MULTIPLE CHOICE: 1) & 3)

1 - A

- Linjärsökning har en tidskomplexitet som är direkt proportionell mot storleken på inmatningen.
- Linjärsökning (linear search) är en enkel algoritm där varje element i en lista eller en sekvens kontrolleras en efter en tills det sökta elementet hittas eller hela listan har genomsökts. Tidskomplexiteten för linjärsökning är **O(n)**, där **n** är antalet element i listan. Detta betyder att tiden det tar att utföra linjärsökningen ökar linjärt med antalet element i listan.
- I motsats till detta har binärsökning (binary search), en tidskomplexitet på **O(log n),** där **n** är antalet element i en sorterad lista. Detta är snabbare än linjärsökning och är möjligt eftersom binärsökning halverar antalet kvarvarande element att söka i vid varje steg.
- Alternativ c) merge sort och d) selection sort är sorteringsalgoritmer som har bättre prestanda än linjärsökning och kräver inte tid som är direkt proportionell mot storleken på inmatningen.

3 - C

- Alternativ "c) an empty sequence" är det rätta svaret eftersom det ofta används som basfall i rekursion på sekvenser av flera skäl:
- Logik och Slutvillkor: När rekursion används för att behandla sekvenser, kan man nå en punkt där sekvensen är tom (dvs. inga element återstår). Att definiera detta som basfallet hjälper till att avsluta rekursionen och undvika oändliga loopar.
- Rekursiva Anrop och Minskning av Problem: Genom att bryta ner sekvensen i mindre delar i varje rekursivt anrop, når man slutligen en punkt där sekvensen är tom. Detta

är liknande hur i en söndra & erövra strategi, problem delas upp i mindre delproblem tills de når en trivial nivå.

- Exempel som Fibonacci-sekvensen: I vissa fall, som när man beräknar Fibonacci-sekvensen, kan ett tomt fall vara naturligt. Till exempel, om man försöker beräkna Fibonacci-talet för index 0 eller 1, skulle dessa vara basfall där man direkt returnerar respektive värde.