D0012E Laboration 2 del 1

Grupp 27
Magda Larsson Andersson, magany-7@student.ltu.se
Andreas Form, forane-9@student.ltu.se
Marcus Asplund, marasp-9@student.ltu.se

December 2020

1 Introduction

Syftet med uppgiften var att lära oss mer om olika Divide and Conquer algoritmer genom att lösa olika problem med målet att uppnå en vis tids komplexitet. Uppgifterna var att hitta de tre minsta talen i en lista och hitta maximala kvoten i en lista. Vi hittade lösningar både med en tids komplexitet på $O(n)$ och $\theta(n \cdot \log(n))$

2 Deluppgift 1

2.1 Incremental approach

Inkrementell är en term som betyder att någonting händer stegvis. I vår programmering gör vi så att vi kollar stegvis av en lista och har skapat en inkrementell algoritm. Uppgiften var att hitta de 3 minsta elementen i en lista med storlek n. Vi skulle också uppnå en tids komplexitet som var O(n)

2.1.1 Design av algoritm

1. Först skapar vi en lista som heter minList där vi har tre platser och är fyllda med oändligt stora tal för att alltid se till att listans värden är mindre

2. Sedan kollar vi på första talet i listan och jämför talet med dem talen som vi har i minListan, då första talet kommer vara mindre än talet som befinner där kommer den att ta dess plats. Vi kollar sedan igenom alla tal i listan och jämför dessa med minListan för att få reda på vilja 3 minsta talen är.

3. När vi har kollat igenom alla tal i listan ska minList vara dem tre minsta talen i rätt ordningen och är därför bara att returna/printa listan.

2.1.2 Tidsanalys

Figure 1: Incremental approach $O(n)$

När man går igenom funktionen som bygger på den Incremental approachen börjar vi kolla på basfallet där man 3 element i listan tar 3 operationer $O(3)$. Sedan kollar vi på hur den påverkas när listan har mer än 3 element.

Då vi koller igenom listan så kommer vi för varje loop titta på nästa värde och köra 5 operationer för att hitta dem 3 största talen. Detta tar $5n$ operationer. När vi sedan kollat igenom alla n tal så kommer vi returna värdet på en operation som tillsammans blir 5n + 1 som i enligt Big O blir $O(n)$

$$\begin{cases} W(1) = 1 \\ W(n) = 5 + W(n-1), \quad n > 3 \end{cases} \tag{1}$$

$$\begin{aligned} W(n) &= (**) \\ 5 + W(n-1) &= (**) \\ 5 + (5 + W(n-2)) = 10 + W(n-2) &= (**) \\ 5k + T(n-k) = .. = 5n + T(1) &= (**) \\ 5n + 1 &= (**) \\ W(n) &= O(n) \end{aligned}$$

2.2 Divide and conquer

Med Divide and conquer så delar man upp listan till mindre "problem" som oftast består av att dela up en lista tills listan blir den önskade storleken. Vi använde Divide and Conquer för att lösa vårt uppgift som var att hitta minsta elementet i en lista som är n stor. Vi hade också ett krav att tids komplexiteten skulle bli $O(n)$

2.2.1 Design av algoritm

1. Listan delas upp rekurisvt på hälften tills den når 3 i list längd och returnerar listan
2. Två olika listor av 3 element får vi av de två rekursiva kallningar. Vi sätter ihop de två listorna till en lista med 6 element
3. Vi kör en min() funktion på listan för att få ut minsta elementet. Elementet tas bort sedan från orginella listan. Detta upprepas 3 gånger tills vi har fått de 3 minsta
4. Listan med de 3 minsta returneras och rekrusionen förstätter tills det bara finns en lista kvar med de 3 minsta

2.2.2 Tidsanalys

För att räkna ut hur mycket tid det tar att få ut dem 3 minsta talen ur en lista med hjälp av divide and conquer räknar vi först hur många gånger som listan kommer dela sig. Listan kommer dela sig $(n/3) - 1$ gånger då listan följer formeln $3 \cdot 2^{k-1}$ för alla positiva heltal på k.

Sedan kommer vi efter alla rekusioner köra min-funktionen som är inbyggd i python $n/3$ gånger och på varje lista som är 3 långa som blir n totalt.

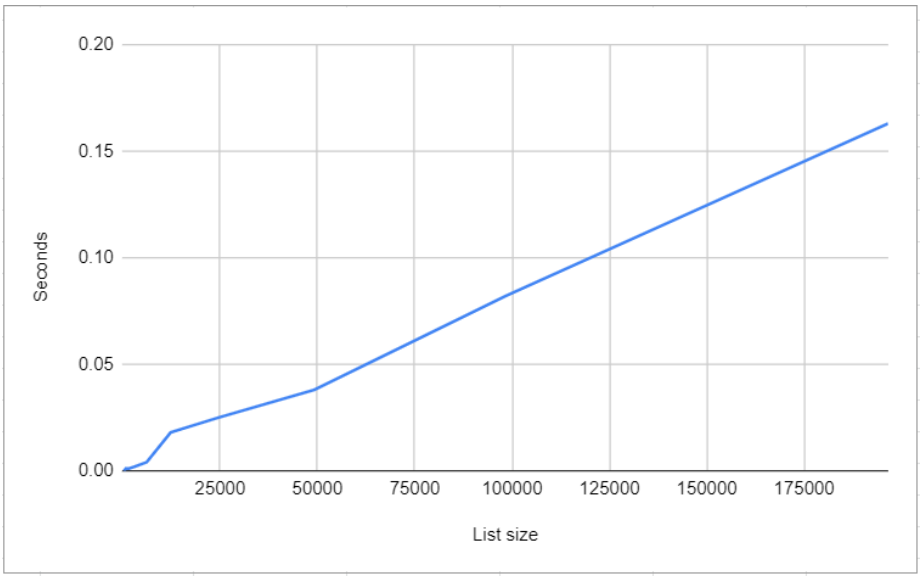

Figure 2: Divide and conquer $O(n)$

$$\begin{cases} T(3) = 1 \\ T(n) = 2T(n/2) + 3n, \quad n > 1 \end{cases} \tag{2}$$

$$\begin{aligned} T(n) &= (**) \\ 2(n/2) + n - 1 &= (**) \\ 2 \cdot (2 \cdot (T(n/4)) + n - 1 = 4(n/4) + n - 1 &= (**) \\ 2^k(T(n/2^k)) + n - 1 &= (**) \end{aligned}$$

$$\begin{aligned} k &= log(n) \\ 2^{\log n} T(n/2^{\log(n)}) + n - 1 &= (**) \\ n \cdot T(1) + n - 1 = 2n - 1 &= (**) \end{aligned}$$

$$T(n) = \theta(n)$$

3 Deluppgift 2

Andra uppgiften bestod av att hitta maximala kvoten på listan $A =< a_1, a_2, ...a_n >$ då $\frac{a_j}{a_i} | 1 \leq i \leq j \leq n$. Två algoritmer skulle designas som respektive hade $O(n \log(n)$ och $O(n)$ tids komplexitet.

3.1 Första versionen, $O(n)$

3.1.1 Design av algoritm

1. Listan med storleken N delas med två rekursiva kallningar som tar varsin halva.
2. Listan delas tills den når rekursiva basfallet då listans storlek är 2 och returnerar kvoten, nämnaren och täljaren i en lista.
3. Vi räknar du de olika möjliga kvoterna som vi logiskt kan räkna ut då nämnaren är till väsnter om täljaren. Vi kollar också på de andra som vi inte kan logiskt se om de är till vänster eller till höger om varandra.
4. Vi returnerar också största täljare och minsta nämnare till nästa rekrusions cykel.

3.1.2 Tidsanalys

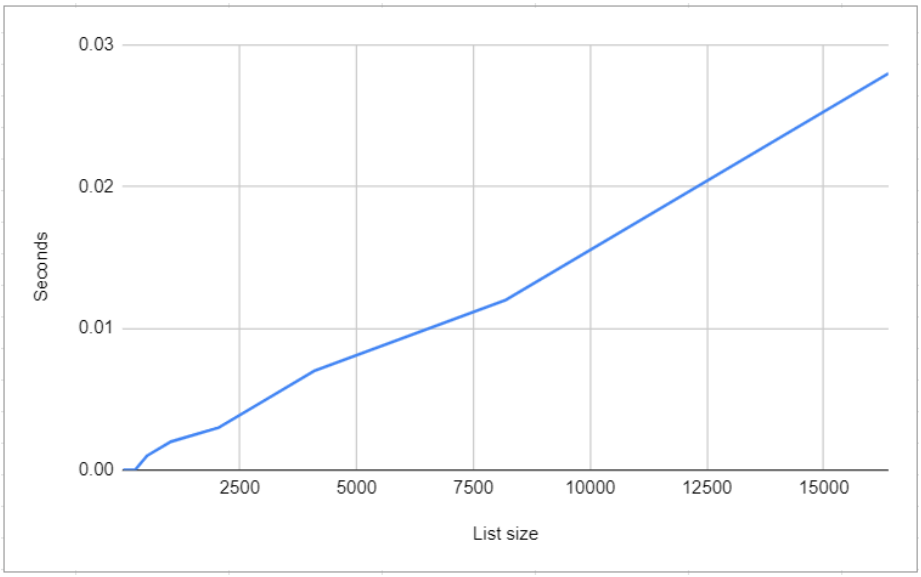

Figure 3: Divide and conquer $O(n)$

$$\begin{cases} T(2) = 1 \\ T(n) = 2T(n/2) + \theta(1), \quad n > 2 \end{cases} \tag{3}$$

$$\begin{aligned} T(n) &= (**) \\ 2(n/2) + 1 &= (**) \\ 2 \cdot (2 \cdot (T(n/4) + 1/2) + 1 = 4(n/4) + 2 &= (**) \\ 2^k(T(n/2^k)) + k &= (**) \end{aligned}$$

$$\begin{aligned} k &= log(n) \\ 2^{\log n} T(n/2^{\log(n)}) + \log(n) &= (**) \\ n \cdot T(1) + \log(n) = n + \log(n) &= (**) \end{aligned}$$

$$T(n) = \theta(n)$$

3.2 Andra versionen $\theta(n \log(n))$

I denna är principen nästan likadan som första versionen utan att den blev långsammare, vilket var poängen, då tis komplexiteten blev $O(n \log(n)$. Den blev långsammare då vi kollade igenom hela listan vid varje rekursion.

3.2.1 Design av algoritm

1. Först kollar vi på om listan som ligger i fokus har en längd av två eller inte, om den har det så returnar vi värdet på andra genom första elementet i listan. Detta är vårt basfall.

2. Om Basfallet inte är uppnått så delar vi listan i mitten och kör samma funktion på första samt andra halvan av listan tills basfallet uppnås. Då basfallet uppnås sparar vi leftBest för vänstra bästa och rightBest för högra.

3. När vi delat alla listor tills basfallet är uppnåt på hela listan så kommer vi jämföra "crossBest" där man jämför två listor mot varandra och tar ut största kvoten från dem.

4. Då svaret antingen har båda siffrorna i vänstra sidan, båda i högra sidan eller en i varje så har vi kollat alla möjliga fall. Där man kan uppnå maxkvot.

3.2.2 Tidsanalys

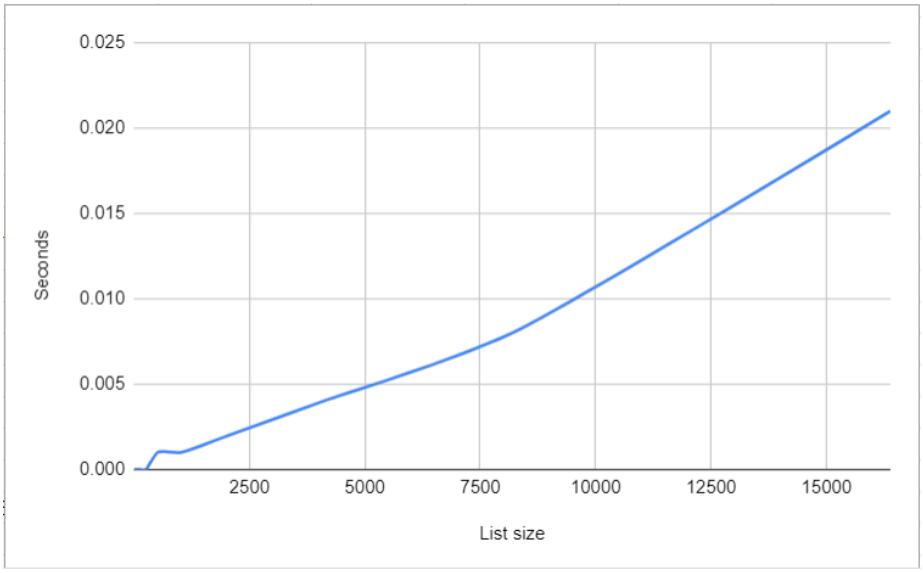

Figure 4: Divide and conquer $\theta(nlogn)$

För att få redo på theta så är det lättaste att försäkra sig om att bästa fallet och sämsta fallet är den samma. Det betyder att theta och Big O kommer vara den samma.

För att räkna ut theta på andra funktionen i uppgift 2 så kollar man först på hur problemet ser ut. Om man kollar på ekvation4 ser man att basfallelt i detta fallet är när man ha ett tal i listan. Det kommer då gå klar på 1 operation. Däremot om man kollar på när listan är längre än 1 tal så kommer man börja dela listan i två delar för att minska problemet i mindre småproblem. Detta är tankesättet bakom Divide and conquer.

Vi kommer där med dela upp listan i hälften tills listan är uppdelad i lika många listor som första listan hade siffror. Detta kommer att göra $O(Logn)$ gånger som man kan se i bilden 5, för varje delning så tittar vi även på den nuvarande listan man arbetar med och tar ut det största och det minsta värdet från listan. Detta görs med $O(n)$, så om man lägger ihop dessa då vi delar hela listan och för varje delning kollar efter största och minsta värdet så kommer vi få ut en tidskomplexitet på $O(nLogn)$.

För att mattematiskt lösa detta problem kan man även kolla på uträkningen under ekvationen 4, där man kan följa hela lösningen av tidkomplexiteten.

$$\begin{cases} T(1) = 1 \\ T(n) = 2T(n/2) + O(n), \quad n > 1 \end{cases} \tag{4}$$

$$\begin{aligned} T(n) &= (**) \\ 2(n/2) + n &= (**) \\ 2 \cdot (2 \cdot (T(n/4) + n/2) + n = 4(n/4) + 2n &= (**) \\ 2^k(T(n/2^k)) + kn &= (**) \end{aligned}$$

$$\begin{aligned} k &= log(n) \\ 2^{\log n} T(n/2^{\log(n)}) + n\log(n) &= (**) \\ n \cdot T(1) + n\log(n) = n + n\log(n) &= (**) \end{aligned}$$

$$T(n) = \theta(n\log(n))$$

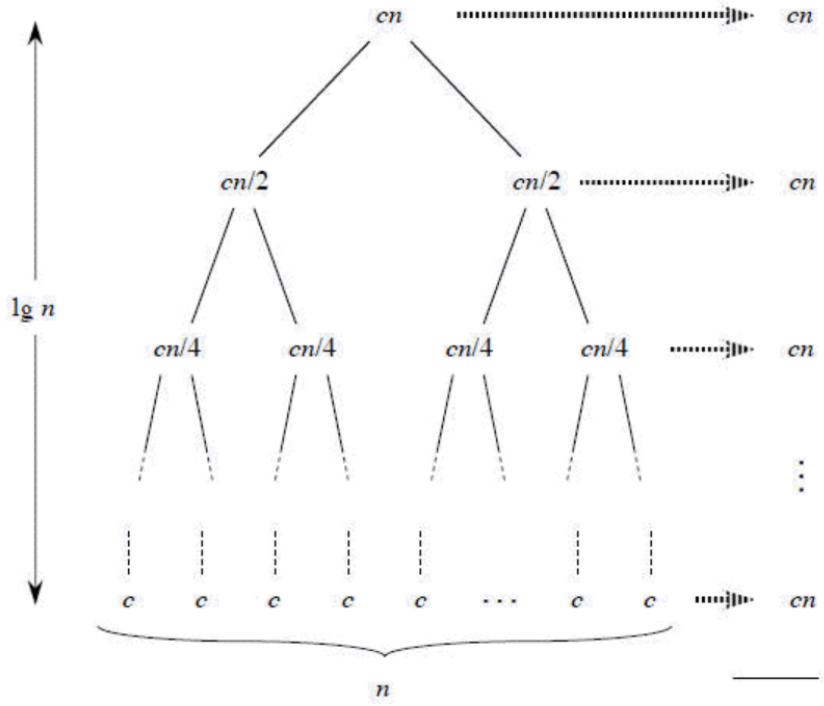

Figure 5: Illustration av divide and conquer