Porteføljeopgave 1

Algoritmer og datastrukturer

2024

Rikke Aa. Boysen

29. Oktober

1 Opgave 1

I denne opgave skulle der skrives en algoritme som tager et naturligt tal som indput og retunere summen af de ulligetal kvadreret fra 1 og til N.

Methoden blev kaldt: $intUnevenSquares()$

Denne tjekker først om n er forskellig fra 0. Hvis dette er tilfældet, tjekkes der her efter om n er et ulige tal. Dette gøres ved at tjekke om $n\%2$ er forskellig fra 0. Herefter retunere funktionen $n * n + UnevenSquares(n-1)$. Dette gøres for at n hele tiden går videre til det næste tal. Hvis n derimod ikke er ulige, retunerer funktionen kun $UnevenSquares(n-1)$ for at n går vidre til det næste tal i rækken.

Koden kan ses i opg1.cpp i den vedlagte zipfil.

2 Opgave 2

For at beregne store-O for algoritmen så kigges der først på de 3 nestede for løkker. Den første løkke løber fra 0 til N, men indeholder $i+=i$, der gør at i bliver fordoblet for hver gang, og fungrer derfor med logaritmisk tid, da den skal bruge halvt så mange inetaritioner på at nå N. Derfor vil denne løkke have $O(LogN)$ som værdi.

Den næste løkke fungrer på samme måde som ovestående, her har den bare $j* = 2$, hvilket også fordobler j for hver gang. Derfor vil denne også have værdien $O(LogN)$.

Den tredje løkke løber fra 0 til $N * \sqrt{N}$ og vil derfor have værdien $O(N * \sqrt{N})$.

Lægges disse tre løkker sammen får man $LogN * LogN * N * \sqrt{N} = Log(N)^2 * N * \sqrt{N} \implies O(Log(N)^2 * N * \sqrt{N})$

Kigger man efterefter på den forløkke der kommer efter de 3 andre, så løber den fra 0 til N*N. Dette giver den en store O på $O(N^2)$

For at afgøre hvilken store O der skal bruges gåes der efter worst case scenario. Eftersom N stiger mere ved N^2 end ved $Log(N)^2 * N * \sqrt{N}$. Så vil store-O for algoritmen være $O(N^2)$

3 Opgave 3

I denne opgave skulle der skrives en algoritme der tager en streng af cifre og retunere true hvis der er tre tal efter hinanden, hvor det tredje tal er summen af de to foregående. Signaturen for fuktionen er: $bool\ additive(Strings)$

Først findes de 3 første cifre i serien. Herefter trækkes ASCII-værdien for 0 fra tallenes ASCII-værdi for at få det korresponderende tal.

Herefter tjekker funktionen om det 3 tal er ligmed summen af de to foregående. Hvis dette er tilfældet retuneres true. Hvis dette IKKE er tilfældet kalder funktionen sig selv, med en substring som indput for at flytte videre til det næste tal i rækken. Her vil tallet der før var nr 2 blive til det første tal.

Denne process gentages indtil der enten er fundet et tal hvor summen af de to foregående svare til tallet eller hele strengen af tal har været igennem.

Koden kan findes i opg3.cpp i den vedlagte zipfil.

4 Opgave 4

Jeg formåede ikke at lave opgaven.

Tanker omkring hvad man kunne have gjort:

Ideen var at jeg ville lave en funktion som først tjekkede summen af 3 tal for hele arrayet. Herefter ville jeg tage denne sum og finde modulo 2. Summene skulle herefter sorteres efter den laveste modulo værdi.

5 Opgave 5

For at finde Store-O tidskompleksiteten for fuktionen startes der ved første for-løkke. Denne løber fra 1 og til $\sqrt{N}$ og vil derfor have en sore-O på $O(\sqrt{N})$.

Herefter kigges der på den næste for løkke. Denne løber fra 1 til N og vil derfor have en store-O på $O(N)$.

Kigger man på sidste forløkke. Så løber denne fra 0 og indtil k bliver større end N. Denne løkke indeholder også linjen $k = k * 2$, der gør at gå bliver fordobelt hver gang. Da dette vil gøre at det tager halv så lang tid at nå N antages det at det er logaritmisk tid. Forløkken vil derfor have en store-O værdi på $O(logN)$

Store O for de tre forløkker samles $Log(N) * N * \sqrt{N}$ og fuktionen vil have en store-O på $O(Log(N) * N * \sqrt{N})$

6 Opgave 6

Denne opgave går ud på at skrive en algortime der retunere summen af heltal mellem 0 og mindre end N, der er dividerbare med 3. Signaturen for algoritmen er $int\ sumDivisibleBy3(intN)$

Koden fungrer ved først at tjekke om N er større end 0. Herefter tjekker funktionen om N er dividerbar med 3. Dette gøres ved at tjekke om $N\%3 = 0$. Hvis dette er tilfældet retunerer

fuktionen $N + sumDivisibleBy3(N-1)$. Der trækkes 1 fra N for at gå videre til næste tal i rækken. Hvis ikke tallet er dividerbart med 3 retunerer funktionen $sumDivisibleBy3(N-1)$ for at gå videre til næste tal i rækken. Dette gøres indtil N bliver 0.

Koden for denne kan findes i opg6.cpp i den vedlagte zipfil.

7 Opgave 7

Jeg formåede ikke at lave opgaven.

8 Opgave 8

I denne opgave skal den nedenstående hash tabel udfyldes via quadratic probing. Den udfyldte del er markeret med rødt. Quadratic probing er en teknink der benyttes til at undgå kollisioner i hashtabellen. Dette gøres ved at tilføje et kvadratisk tal, oftest 2, til det opriendelige index.

Først placeres Q. Da Q hasher til index 7 og denne er ledig indsættes den direkte.

Herefter skal C placeres. Denne hasher til index 8 som allerede er optaget. Derfor tilføjes 1^2. Dette giver $8 + 1^2 = 9$. Index 9 er ledigt og C placeres dermed på denne plads.

Til sidst placeres H. Denne hasher til index 2 der allerede er besat. Dermed udføres overstående igen, $2 + 1^2 = 3$. Da index 3 også er optaget forsøges der nu med 2^2, $2 + 2^2 = 6$. Denne er forsat optaget og tallet hæves igen, $2 + 3^2 = 11$. Da index 11 ikke er en del af tabellen, startes der forfra ved index 0. Denne er ledig og H placeres dermed på denne plads.

0	H
1	
2	V
3	R
4	
5	
6	P
7	Q
8	E
9	C
10	F

9 Opgave 9

For at finde Store-O tidskompleksiteten for denne function, så blev der udført test i VS code. Her blev der indsat et print statement for at se hvad der skete med n når fuktionen kørte. Udfra dette kunne det ses at funktionen kørte indtil n var ligmed eller mindre end 1.

Det kunne også ses at får hver gang n gik en op blev der doobelt så mange prints. fx hvis n = 2, løb funktionen 2 gange igennem, mens hvis n = 3 printer den 4 gange.

Da metoden kalder sig selv to gange og det kunne ses fra testen at gennemgangen afhænger af n, så vil store-O værdi være $O = (2^n)$.

10 Opgave 10

I denne opgave skulle der skrives en rekusiv metode der retunerede totals-logaritmen af N. Signaturen for denne er $int\, logTO(intN)$

Metoden tjekker først om N er forskellig fra 0 og om N er en potens af to. Dette gøres ved at tjekke om $N\%2 = 0$, da enhver potens af 2 også er dividerbar med 2.

Hvis dette er tilfældet retunerer funktione $1 + logTo(N/2)$ Dette gør den for at tælle antallet af gange N kan halveres før den bliver 1, da denne vil svare til potensen 2 skal opløftes i for at blive N.

Koden for denne kan findes i opg10.cpp i den vedlagte zipfil.

11 Opgave 11

I denne opgave skulle der skrives en algoritme, som kaldt med tabellen og eventuelt tabellens længde, kan afgøre om en kandidat fik mere end 50 % af stemmerne. Hvis dette var tilfældet skulle den retuneres deres nummer ellers skal den retunere -1. Signaturen for denne er $intvotes(intarray[], intlen)$.

Algoritmen fungerer ved først at initialisere en tomt array der bruges som tæller.

Herefter kører forløkken igennem hvert element i arrayet hvor hver unikke element tildeles et index. Hvis der er flere af samme nummer øges antallet i "count" tabellen.

Derefter tjekkes der om en kandidat har flere stemmer end 50% af alle stemmerne. Dette gøres ved at tjekke om et index i count er større end halvdelen af længden af hele stemme arrayet.

Koden kan ses i opg11.cpp i den vedlagte zipfil.

11.1 Tidskompleks for algoritmen

For at finde tidskompleksiteten for algoritmen kigges der først på forløkken. Denne løber fra 0 og til længden af arrayet. Dette ville være tilsvarende til 0 og op til N. Dette ville give forløkken en store-O på $O(N)$.

Herefter kigges der på if statementet. Da denne kun retunere værdien, vil store-O være $O(1)$. '

Da dette ikke forlænger tiden vil store-O for hele algoritmen være $O(N)$