Porteføljeopgave 1

Mathias Balling Christiansen

October 28, 2024

Contents

1 Opgave

Jeg har implementeret algoritmen så ved lige tal trækkes 1 fra. Og ved ulige tal trækkes 2 fra for at ramme det næste ulige tal. Derved spares uoverflødige kald. Algoritmen stopper ved n=1.

2 Opgave

Funktionen har et for-loop med 2 nestede for-loops and derefter et sequentielt for-loop. Det yderste for-loop fordobler i hver gang det kører, derfor kan den estimeres med $\log N$. Det midterste for-loop fordobler j hver gang det kører, derfor kan den estimeres med $\log N$. Det inderste inderste for-loop går fra 0 til $N \cdot \sqrt{N}$.

Samlet giver dette: $N^{3/2} \cdot \log(N)^2$.

Efter det nestede for-loop er et for-loop der går til $N \cdot N$. Hvilket er N^2.

Ved at plotte begge ses det at $N^{3/2} \cdot \log(N)^2$ og N^2 mødes ved 65536. Derfor er tidskompleksiteten:

$$N < 65536: \quad O(N^{3/2} \cdot \log(N)^2)$$
$$N > 65536: \quad O(N^2)$$

3 Opgave

Jeg har implementeret algoritmen så den stopper ved at dens længde er under 3 (basecase), da der skal bruges 3 tal til at sammenligne summen af 2 tal med den 3. Jeg tjekker bogstavests talværdi ved at trække ascii værdien af 0 fra. Hvis summen af 2 tal er lig med det 3. tal returneres 'true'. Hvis længden stadig er over 3 og summen ikke gav det 3. tal, kaldes funktionen med samme tekst, men hvor det første tal er fjernet.

4 Opgave

Her implementeres en algoritme der finder hvilke 3 tal i et array, der giver det tætteste tal på en potens af 2. For at optimere den har jeg et tjek der ser om vi rammer en præcis potens af 2. Hvis vi gør det, kan vi ikke komme tættere og derfor returneres. Tidskompleksiteten af algoritmen er $O(n^3)$, på grund af de 3 for-loops. For at få lavere tidskompleksitet algoritmen kunne alle summene af to tals gemmes i et hash map. Herefter kunne der itereres over alle værdier i dette, hvor det sidste tal summere og sammenlignes med potens af 2. Denne algoritme ville have 2 dobbel for-loops i sekvens, hvilket er $O(n^2)$

5 Opgave

Der er 3 for-loops. Det yderste kører $\sqrt{N}$ gange. Det midterste kører N gange. Det inderste vil fordobler k hver gang den kører. Dette svarer til en logoritmisk tidskompleksitet. Til sammen bliver tidskompleksiteten $O(N \cdot \sqrt{N} \cdot \log N)$

6 Opgave

Her implementeres en algoritme der finder summer tal der er dividerbare med 3 i intervallet [0,N]. Hvis tallet er 0 returneres 0 (basecase). Hvis tallet er dividerbart med 3, returneres tallet og vi kalder algoritmen igen med tallet minus 3. Hvis tallet ikke er dividerbart med 3, returneres tallet minus (tallet modulus 3). Ved at trækket tallet modulus 3 fra ved vi at vi rammer et tal der er dividerbart med 3 næste gang.

7 Opgave

Her implementeres en algoritme der finder ud af om et tal, Z, kan beskrives af X^Y. Hvis flere kombinationer er muligt skal største X værdi vælges. Jeg kigger på alle værdier på X og Y, der er over 2 og hvor X^Y er under eller lig med resultatet. Dette gør at der spares mange udregning. Tidskompleksiteten for denne algoritme er $O(N^2)$

8 Opgave

Original:

Index	Værdi
0	
1	
2	V
3	R
4	
5	
6	P
7	
8	E
9	
10	F

Efter Q til indeks 7:

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

Efter C til indeks 8. Indeks 8 er optaget så vi lægger 1^2 til:

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

Efter H til indeks 2. Indeks 2 er optaget så vi lægger 1^2 til, men 3 er også optaget. Så vi lægger 2^2 til, men 6 er også optaget. Så vi lægger 3^2 til, her ender vi på 0:

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

9 Opgave

Denne funktion tager et tal n og kalder funktionen rekursivt med n-1 og n-2. Det vil sige funktionen har tidskompleksiteten $O(2^N)$, fordi den kalder sig selv 2 gang pr. rekursion. Fx n=10 til rekursion kalder n=9 og n=8. n=9 kalder derefter n=8 og n=7. n=8 kalder n=7 og n=6. Så $1 \rightarrow 2 \rightarrow 4 \rightarrow 8$ Dvs der ender med at være $O(2^N)$ tidskompleksitet.

10 Opgave

I denne opgave er mit basecase at $N < 2$. Her returneres 0. For alle tal over returneres 1 plus et rekursivt kald med $N/2$. Derved går vi fra functionskaldene $32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2$ hvilket giver $1 + 1 + 1 + 1 + 1 + 0 = 5$.

11 Opgave

Jeg starter med at lave et tomt hash map. Herefter går jeg igennem stemmerne. En stemme fx 7 er en key og dens værdi er antal gange den key er forkommet. Hvis det første gang vi ser en key sættes værdien til 1 og eller bliver den incremented. Undervejs tjekkes om den indsatte kandidat har over halvdelen af stemmerne og hvis den har det returneres true, da kun en kan have det.

Min tidskompleksitet for algoritmen er O(N) da vi kun går igennem algoritmen n gang.