VT22 DT046G/DT064G Datateknik GR (B)
Andreas Gabrielsson
Anga2001@student.miun.se

Laboration: Analys av sökalgoritmer

Metod

Laborationen utfördes genom att jämföra fyra olika sökningsmetoder: linjär sökning, binär sökning, Hash-tabell och binärt sökträd. Mätningarna baserades på sökningar inom en lista av primtal och tiden det tog för varje sökning sparades i en csv-fil.

Kompilatoroptimeringar

För att förhundra kompilatoroptimeringar (som jag listade ut var ett problem) använda jag mig av nyckelordet ”volatile” med en global variabel i mina mätningar, för att se till att kompilatorn inte optimerade bort sökningar där resultatet inte användes.

Python

Python-skriptet använder matplotlib för att visualisera tidsprestandan för olika sökningsalgoritmer, baserat på datan i CSV-filen.

Översikt av vad koden gör:

1. Läs in CSV-filen och för varje rad, spara datan för vör varje algoritm i en container.
2. Skapa en ny figur med matplotlib.
3. för varje algoritm, plotta ut datan som punkter med felstaplar representerande standardavvikelsen.
4. Använd NumPy för att anpassa en polynom till varje algoritms data och kurvan.
5. Lägg till titel, x och y-axel etiketterna, ”legend” och rutnät i plotten.
6. Spara plotten som en bildfil.

Programkod

Programmet är uppdelat i flera filer och klasser.

- **Main.cpp** är kärnan i programmet där sökningarna utförs och mätningar samlas.
- **Binary_search.cpp, Beinary_search_tree.cpp, Linear_search.cpp** innehåller implementer av sökalgotitmerna.
- **HashTable.cpp/h** innehåller implementationer av en hash-tabell med länkade listor för att hantera kollisioner.
- **File_handling.cpp** hanterar skrivning av resultat till en CSV-fil.
- **Generate_primes.cpp** innehåller funktion för att generera primtal.
- **Measurement.cpp** utför mätningarna och lagrar dem i en ”SearchData”-struktur.
- **Timer.cpp** används för att mäta tiden det tar för varje sökning.

VT22 DT046G/DT064G Datateknik GR (B)
Andreas Gabrielsson
Anga2001@student.miun.se

Algoritmerna

Linjärsökning:

Genomsöker varje element i en lista tills det önskade värdet hittats eller hela listan har genomsökts.
Tänk dig att du letar efter ett specifikt kort i en hög som ligger framför dig på bordet. Du börjar med det översta kortet och bläddrar genom varje kort, ett i taget, tills du hittar det kort du letar efter, eller tills du gått igenom hela högen.

```
int linearSearch(const std::vector<int> &arr, int value) {
    for (int i = 0; i < arr.size(); ++i) {
        if (arr[i] == value)
            return i;
    }
    return -1;
}
```

Binärsökning:

Delar listan i två halvor och bestämmer vilken halva det önskade värdet kan vara i, och upprepar processen tills värdet hittats.
Tänk dig att du har en bok där varje sida representerar ett nummer i stigande ordning. Om du letar efter ett specifikt nummer, börjar du på mitten av boken och tittar på sidnumret. Om sidnumret är för lågt, öppnar du boken på mitten och kollar till höger i bokhalvan, men om det är för högt fokuserar du på den vänstra halvan. Denna process med att halvera boken upprepar du till dess att du hittar rätt sida eller inser att sidan inte finns.

```
int binarySearch(const std::vector<int> &primes, int target_value) {
    int low = 0;
    int high = primes.size() - 1;

    while (low <= high){
        int mid = (low + high) / 2;
        if (primes[mid] == target_value)
            return mid;
        else if (primes[mid] < target_value)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
```

Hash-tabell:

Använder en hashfunktion för att bestämma indexet för varje värde och gör det snabbt att hämta värdet baserat på dess hash.
Tänk dig en stor bokhylla med många fack, där vaje fack har en unik etikett. När du vill hitta en bok i bokhyllan, använder du bokens titel för att räkna ut exakt vilket fack den ska vara i, genom hashfunktionen. Istället för att leta igenom varje fak för att hitta en bok, använder du helt enkelt dess titel för att direkt räkna ut vilket fack den ligger i.

VT22 DT046G/DT064G Datateknik GR (B)
Andreas Gabrielsson
Anga2001@student.miun.se

```
int HashTable::search(int data){
    int hash = data % reserved_size;

    HashTableNode* node = table[hash].get();
    while(node != nullptr){
        if(node->data == data){
            return node->data;
        }
        node = node->next.get();
    }
    return -1;
}
```

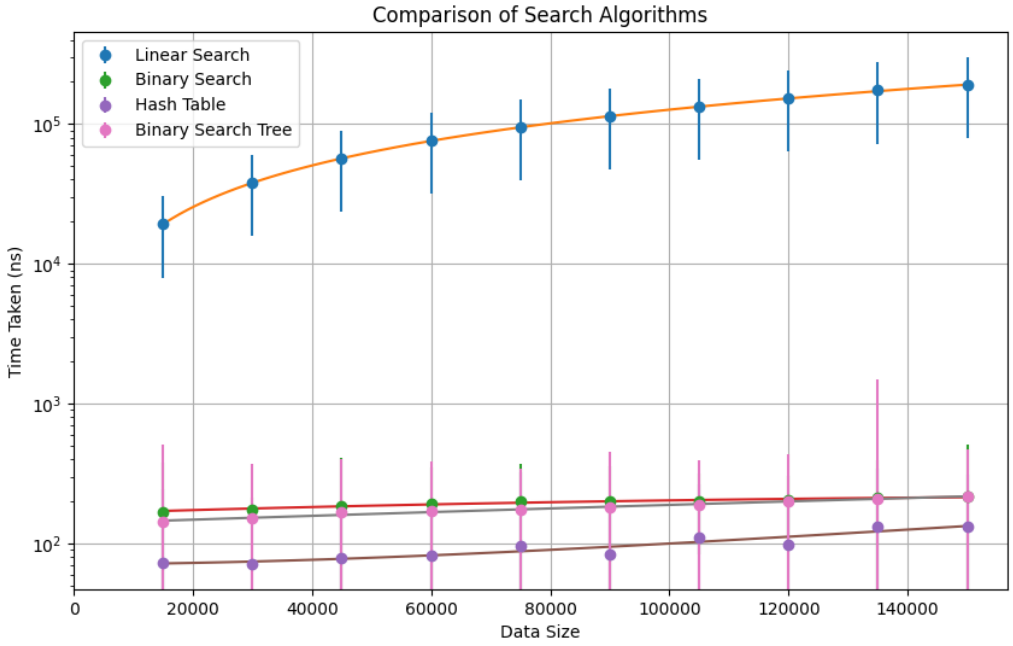
Binärt sökträd:

Använder en trädstruktur där varje nod har högst två barn. Värdet i varje nod bestämmer vilket av dess barn som kan innehålla det önskade värdet.
Tänk dig ett familjeträd där varje person har två barn. Varje person i trädet har ett nummer på sig. Om du då letar efter ett specifikt nummer börjar du med roten (personen i toppen). Om numret du letar efter är mindre än rotnumret, går du till vänster barn, men om det är större, går du till höger barn. Detta upprepar du, med att antingen gå till vänster eller höger, tills du hittar rätt person eller inser att personen inte finns i trädet.

```
BinarySearchTreeNode* BinarySearchTree::search(int value, BinarySearchTreeNode *node) const {
    if (node == nullptr || node->data == value) {
        return node;
    }
    if (value < node->data) {
        return search(value, node: node->left);
    }
    return search(value, node: node->right);
}
```

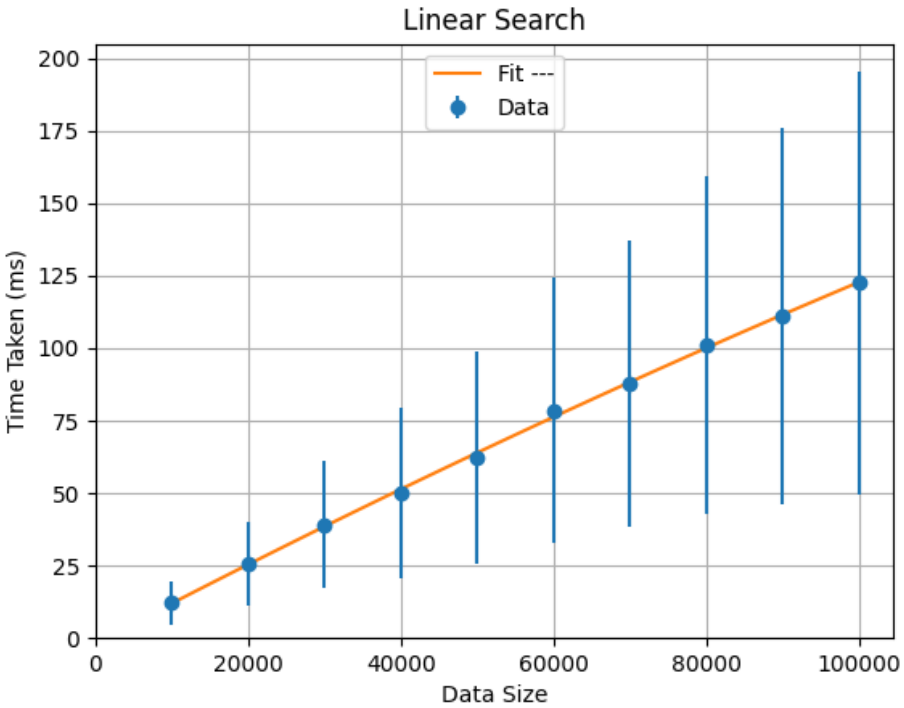
VT22 DT046G/DT064G Datateknik GR (B)
Andreas Gabrielsson
Anga2001@student.miun.se

Resultat och grafer



Linjärsökning:

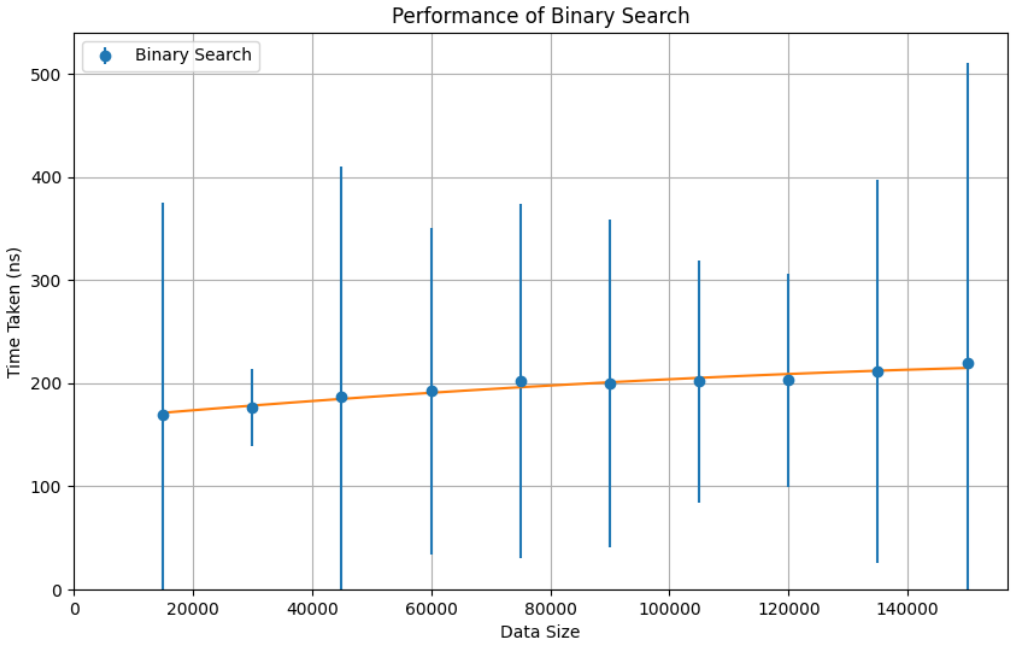
Linjärsökningens prestanda ökar linjärt med datamängden, vilket stämmer överens med dess tidskomplexitet av O(n). Detta beror på att varje element i datamängden behöver inspekteras tills det önskade värdet hittats eller hela listan är genomsökt. När datamängden ökar krävs det i genomsnitt fler steg för att hitta det önskade värdet. [1]



VT22 DT046G/DT064G Datateknik GR (B)
Andreas Gabrielsson
Anga2001@student.miun.se

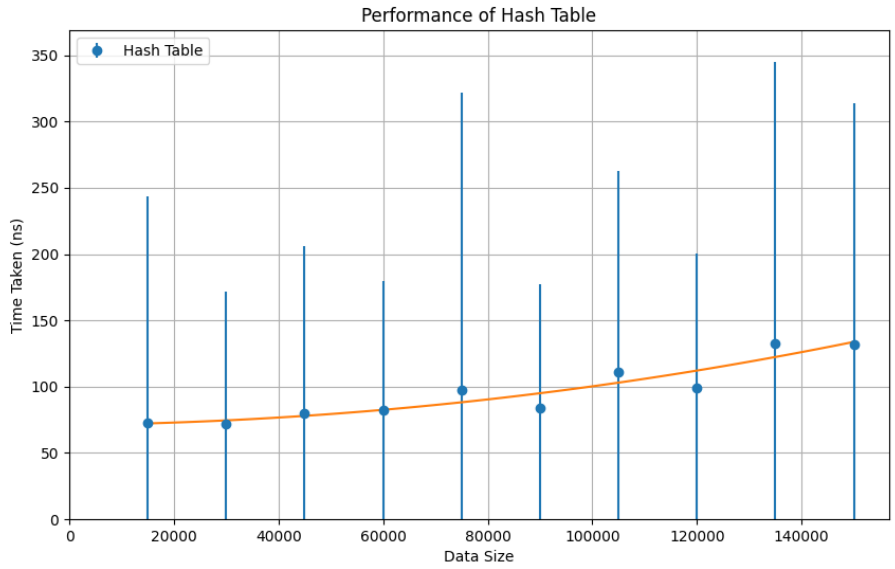
Binärsökning:

Binärsökning visade en mycket snabbare prestanda i jämförelse med linjärsökningen och dess prestanda ökar logaritmiskt med datamängden, vilket stämmer överens med sin tidskomplexitet av O(log n). Detta beror på att den effektivt delar datamängden i halvor vid varje steg, vilket kraftigt minskar antalet element som måste inspekteras! [2]



Hash-tabell:

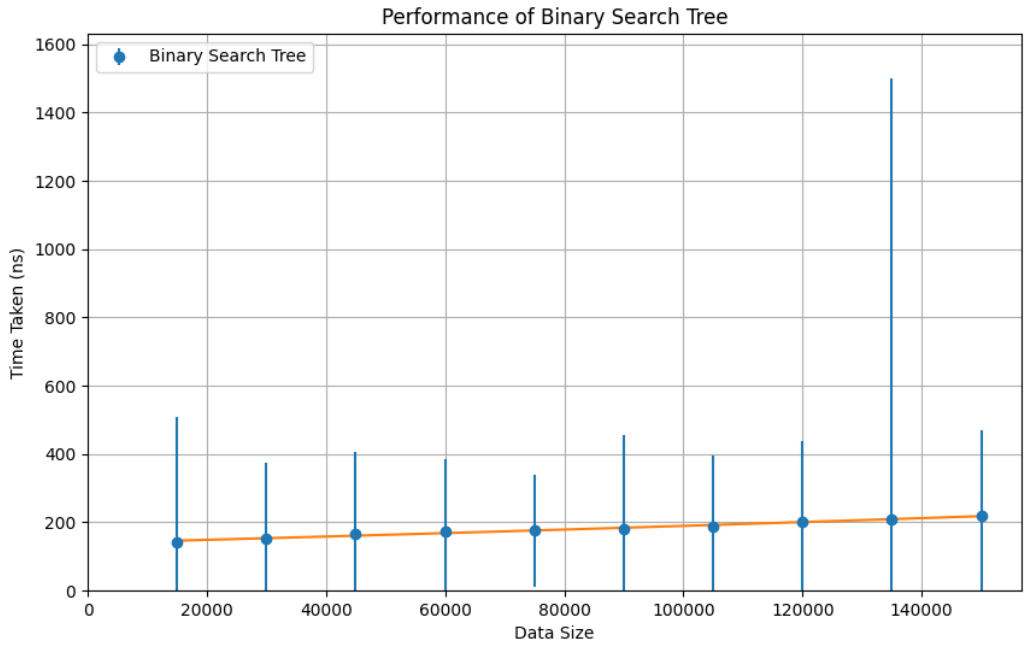
Hash-tabellen visade sig vara mycket effektiv för stora datamängder, med tider som ligger nära konstanta oavsett datamängden. Detta stämmer överens med dess förväntade tidskomplexitet av O(1) för både bästa och genomsnittliga fall. Dock, i sämsta fall, om det finns många kollisioner, kan tidskomplexiteten öka. [3] [4]



VT22 DT046G/DT064G Datateknik GR (B)
Andreas Gabrielsson
Anga2001@student.miun.se

Binärt sökträd

Binärt sökträd visade en konsekvent prestanda över de olika datamängderna, vilket förväntas då trädet nu är balanserat. Tidskomplexiteten för ett balanserat binärt sökträd förväntas ligga på O(log n) i genomsnitt, vilket kan ses i mitt mätresultat. [5]



Diskussion/slutsats

I denna laboration utfördes en analys av olika sökalgoritmer för att förstå hur de presterar. De experimentella resultaten stämmer överens med de teoretiska förväntningarna jag studerat på internet.

Binärsökningen bekräftade att den är mer effektiv än linjärsökning med logaritmisk tidskomplexitet O(log n), särskilt när datamängderna ökar.

Hash-tabellen var den algoritm som var svårast att förstå hur den fungerar, men efter en del tittande på Youtube fick jag grepp om den. Den visade sig vara en effektiv metod för att hitta specifika värden i en datamängd. Själva nyckeln till hash-tabellens effektivitet ligger alltså i att omvandla värden till unika index, vilket gör att åtkomsttiden i genomsnitt blir O(1).

Programkod

https://github.com/anga95/doa_labb3.git

Bibliography

[1] Wikipedia, "Linear Search," [Online]. Available: https://en.wikipedia.org/wiki/Linear_search. [Accessed 21 08 2023].

VT22 DT046G/DT064G Datateknik GR (B)
Andreas Gabrielsson
Anga2001@student.miun.se

[2] Wikipedia, "Binary search algoritm," [Online]. Available: https://en.wikipedia.org/wiki/Binary_search_algorithm. [Accessed 21 08 2023].

[3] HackerRank, "Data Structures: Hash Tables," [Online]. Available: https://youtu.be/shs0KM3wKv8. [Accessed 21 08 2023].

[4] wikipedia, "Hash table," [Online]. Available: https://en.wikipedia.org/wiki/Hash_table. [Accessed 21 08 2023].

[5] wikipedia, "binary search tree," [Online]. Available: https://en.wikipedia.org/wiki/Binary_search_tree. [Accessed 21 08 2023].