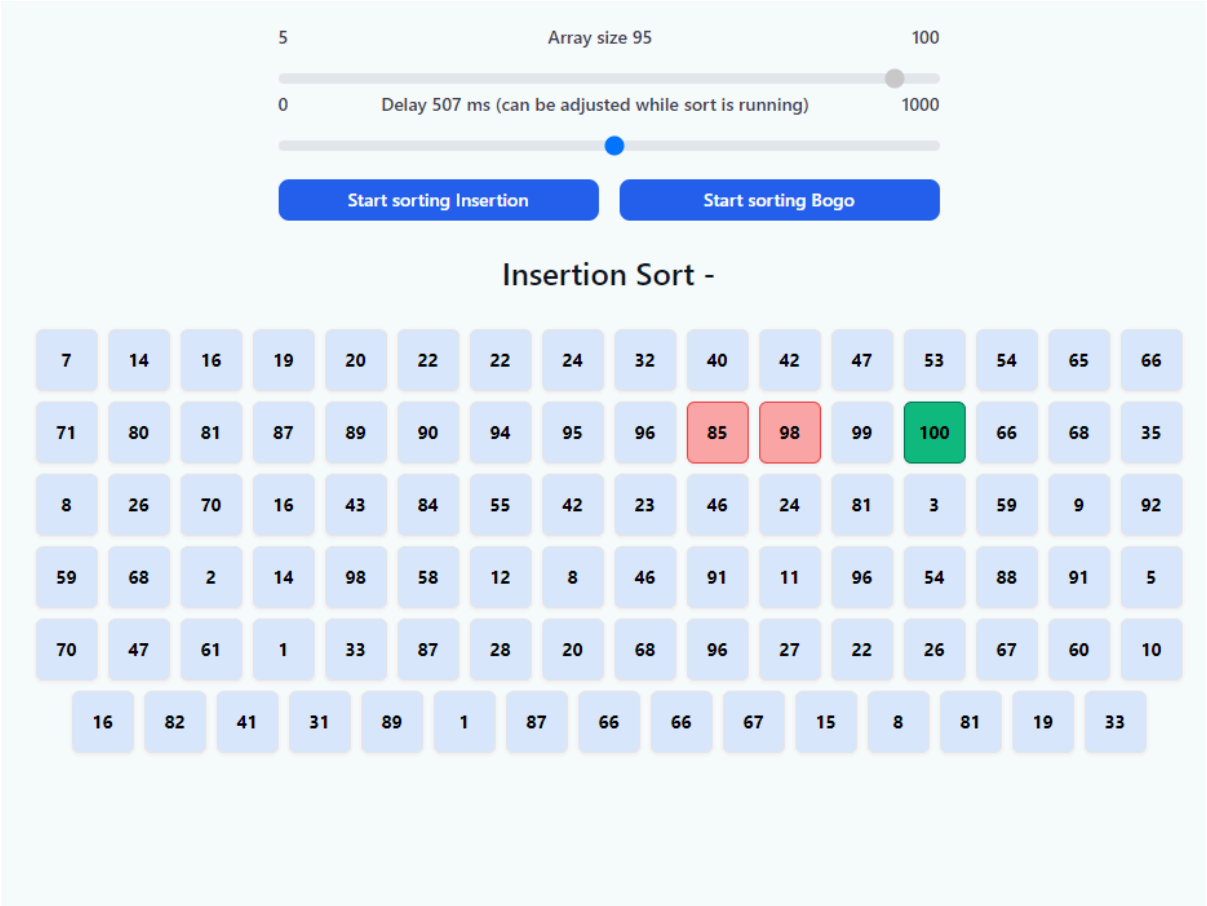
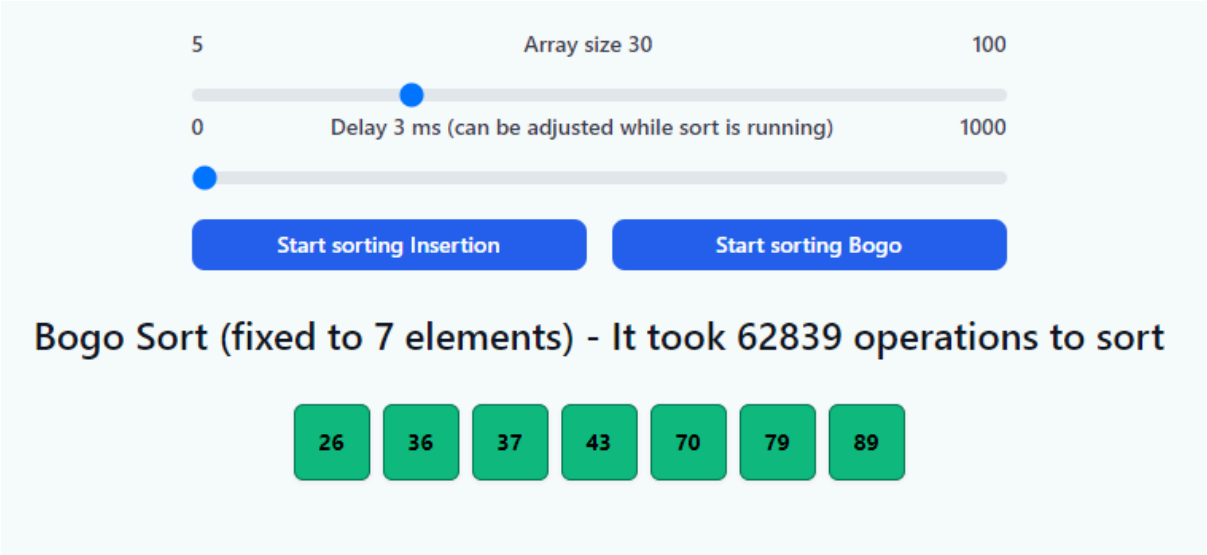
Navn: Frederik Hallengreen Hansen

DSA midtvejsprojekt: SortWizard: Interactive Sorting View





Github: https://github.com/FHallengreen/MidWayProject-DSA

Navn: Frederik Hallengreen Hansen

SortWizard: Interactive Sorting View

Jeg har lavet en visualizer af Insertion Sort og Bogo sorteringsalgoritmerne.
Programmet fungerer ved at man i starten vælger størrelsen på listen af tal og derefter hvor hurtigt programmet skal køre. Derefter vælger man hvilken algoritme man vil have visualiseret. Når man klikker start, vil den oprette et array af tilfældige tal, som herefter vil blive sorteret efter enten insertion sort eller bogo sort algoritmen. Imens programmet kører, kan man ændre hastigheden ved brug af slideren i toppen, som man kan indstille til en pause på 0 ms eller op til et sekund mellem hver operation.

Algoritmer

Insertion sort:

Insertion Sort er en intuitiv og enkel sorteringsalgoritme, der bygger på princippet om at opbygge et sorteret under array ét element ad gangen, meget ligesom den måde man ville sortere kort i hånden. Den gennemgår arrayet og placerer hvert nyt element på den rette plads i det allerede sorteret segment af arrayet. Dette gøres ved at vælge et 'nøgleelement' og sammenligne det baglæns med allerede sorteret elementer, indtil den rette plads for nøgleelementet findes. Processen gentages for hvert element i arrayet, indtil hele arrayet er sorteret.

1. **Loop** Den ene mulighed er insertionsort og den fungerer som følger: En løkke som starter fra det andet element i arrayet og bevæger sig fremad.
2. **Nøgleelement**: Det aktuelle element (key = arr[i]) som er det, der skal placeres korrekt i arrayet.
3. **While loop**: Sammenligner key med hvert element til venstre for det (startende med j = i - 1) og flytter hvert af disse elementer ét skridt til højre, hvis de er større end key.

- **Visualisering**: displayArray(arr, [j, j + 1], i); viser den aktuelle tilstand af arrayet og markerer de elementer, der sammenlignes.
- **Forsinkelse**: await new Promise(resolve => setTimeout(resolve, (delay / 2))); introducerer en forsinkelse baseret på brugerindstillingen, så man kan observere hvert trin.

4. **Indsættelse**: Når den korrekte position for key er fundet, indsættes den ved at sætte arr[j + 1] = key;.
5. **Operationstæller**: Inkrementerer operations for hver sammenligning og flytning, hvilket giver en ide om, hvor mange trin algoritmen har taget.
6. **Afsluttende visualisering**: Når arrayet er fuldt sorteret, vises den endelige tilstand gennem displayArray(arr);.

- **Resultatvisning**:

En tekst vises: "Took ${operations} operations to finish", som informerer brugeren om det samlede antal operationer, der blev udført for at fuldføre sorteringen.

- **Hastighedsjustering**:

Brugeren kan til enhver tid under kørslen justere hastigheden ved hjælp af en skyder. Dette justerer forsinkelsen mellem hver operation, fra 0 ms (ingen forsinkelse, maksimal hastighed) til 1 sekund (langsom visualisering, nem at følge).

Github: https://github.com/FHallengreen/MidWayProject-DSA

Navn: Frederik Hallengreen Hansen

Insertion Sorts effektivitet, målt ved tidskompleksitet, varierer baseret arrayet. I det bedste tilfælde, hvor arrayet allerede er sorteret (eller næsten sorteret), opererer Insertion Sort med en tidskompleksitet på O(n). Dette scenario opstår, fordi algoritmen kun behøver at gennemgå arrayet én gang for at bekræfte, at hvert element allerede er på sin rette plads, hvilket kræver minimal indsats med kun én sammenligning per element.
I det værste tilfælde, hvor arrayet er sorteret i omvendt rækkefølge, har Insertion Sort en tidskompleksitet på O(n^2). Dette skyldes, at hvert element i arrayet potentielt skal sammenlignes med alle forudgående elementer, før det finder sin korrekte placering. For hvert nyt element, algoritmen evaluerer, øges antallet af nødvendige sammenligninger og flytninger, hvilket resulterer i en kvadratisk stigning i antallet af operationer.

```
4   export async function insertionSort(arr) {
5       let operations = 0;
6       for (let i = 1; i < arr.length; i++) {
7           let key = arr[i];
8           let j = i - 1;
9           while (j >= 0 && arr[j] > key) {
10              displayArray(arr, [j, j + 1], i);
11              await new Promise(resolve => setTimeout(resolve, (delay / 2)));
12              arr[j + 1] = arr[j];
13              arr[j] = key;
14              displayArray(arr, [j, j + 1], i);
15              await new Promise(resolve => setTimeout(resolve, delay));
16              j--;
17              operations++;
18          }
19          arr[j + 1] = key;
20          operations++;
21      }
22      displayArray(arr);
23      document.getElementById("dynamic-text").textContent = `Took ${operations} operations to finish`;
24  }
```

Bogo Sort:

Bogo sort bliver også kaldt dum sortering, da det er op til ren tilfældighed at sortere den rigtigt.I bedste tilfælde har den en O tid på O(n), hvilket betyder at arrayet tilfældig er sorteret rigtigt fra start af. Gennemsnitligt er O tiden dog O((n+1)!), hvilket giver nogle utroligt høje tal hvis der er mange tal i arrayet. Jeg har derfor i min løsning sat den fast til 7, da det vil give en O tid på O(7+1)!) = O(8!) = 8x7x6x5x4x3x2x1 = 40320 operationer i snit for at sortere arrayet korrekt. Med et lille array giver dette ingen problemer, men med et større array som f.eks. 10, betyder det at der i snit skal bruges omkring 3.6 millioner operationer for at sortere arrayet.

Algoritmen fungerer som følger:

1. **While loop**: Der kører et while loop, som kalder to funktioner indtil isSorted bliver sat til true.
2. **Shuffle:** Arrayet bliver tilfældigt shufflet ved brug af Fisher-Yates algoritmen. Det er denne shuffle som er grunden til bogo sort er så ineffektiv, da tallene bare sættes tilfældigt.
    2.1. Starter i sidste element i arrayet og bevæger os mod første element.
    2.2. Vælger et tilfældigt index j mellem 0 og i.

Github: https://github.com/FHallengreen/MidWayProject-DSA

2.3. Elementerne byttes ved at oprette en midlertidig variable til at opbevare værdien af arr[i]. Derefter sætter arr[i] til arr[j] og til sidst bliver arr[j] sat til i som blev gemt i variablen.

2.4. Processen gentages indtil man når det andet element i arrayet, da der ikke er behov for at bytte det første element med sig selv.

3. **Tjek om arrayet er sorteret**.

3.1. **Loop**: Vi laver et loop fra andet element i arrayet og viser dette til brugeren.

3.2. **hvis falsk:** Hvis værdien af arr[i] er mindre end det forrige element, bliver der returneret false, indekset opdateret og visualiseringen "nulstillet".

3.3. **hvis sandt**: Hvis den når til vejs ende og dermed konkluderer alle elementer er sorteret korrekt, bliver visningen opdateret ved at gøre hele arrayet grønt og returnere sandt.

```
async function bogoShuffle(array) {
    for (var i = array.length - 1; i > 0; i--) {
        var j = Math.floor(Math.random() * (i + 1));
        var temp = array[i];
        array[i] = array[j];
        array[j] = temp;
        operations++;
    }
}

async function isBogoSorted(array) {
    let sortedUpToIndex = null;

    for (let i = 1; i < array.length; i++) {
        displayArray(array, [], null, i - 1)
        if (array[i] < array[i - 1]) {
            sortedUpToIndex = null;
            displayArray(array, [], null, sortedUpToIndex)
            return false;
        }
        operations++;
        await new Promise(resolve => setTimeout(resolve, delay));
    }

    displayArray(array, [], null, array.length - 1)
    return true;
}


export async function bogoSort(arr) {
    operations = 0;
    let isSorted = false;
    while (!isSorted) {
        await bogoShuffle(arr);
        isSorted = await isBogoSorted(arr);
    }
    document.getElementById("dynamic-text").textContent = `It took ${operations} operations to sort`;
}
```

Inspiration:

Inspirationen til at lave denne interaktive sorterings visning, kommer fra forrige opgave med labyrinter, hvis der også blev visualiseret for hvad step hvad der skulle ske. Jeg synes det kunne være sjovt at vise hvert step i processen for bedre at forstå hvordan algoritmen fungerer ved at kunne justere hastigheden selv. Jeg kiggede på [denne Big O visualiser](#) hjemmeside for at finde inspiration til hvilken algo jeg ville lave. Efter længere søgen stødte jeg på Bogo sort og synes den kunne være sjov at visualisere sammen med Insertion sort. Flere links til resourcer findes i README filen.

Github: https://github.com/FHallengreen/MidWayProject-DSA