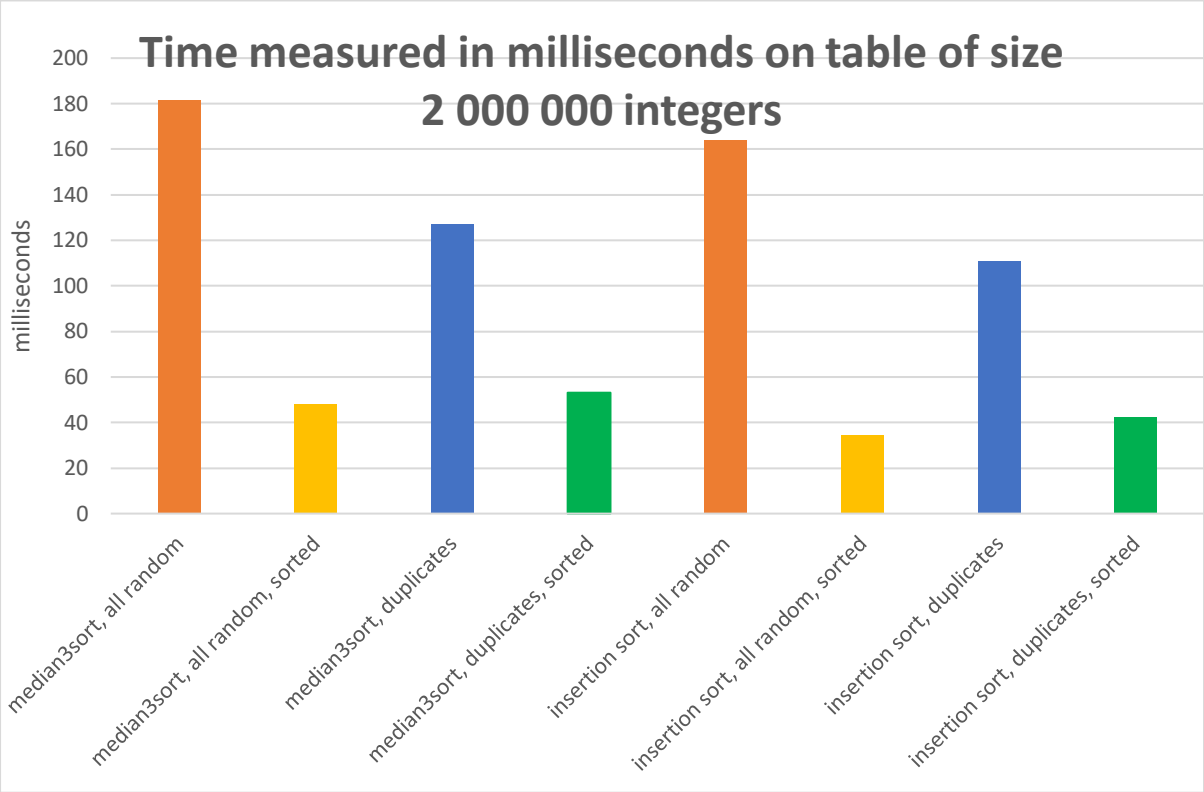
Joachim Grimen Westgaard

# Rapport – øving 3
# Oppgave 1 - Quicksort med hjelpealgoritmer

Jeg valgte å se på bruken av innsettingssortering som hjelpemetode for quicksort-metoden. Ved hjelp av testing og feiling, fant jeg fram til en størrelse på deltabellene hvor quicksort med innsettingssortering ble raskere enn ordinær quicksort med mediansortering ved deltabellengde 3.

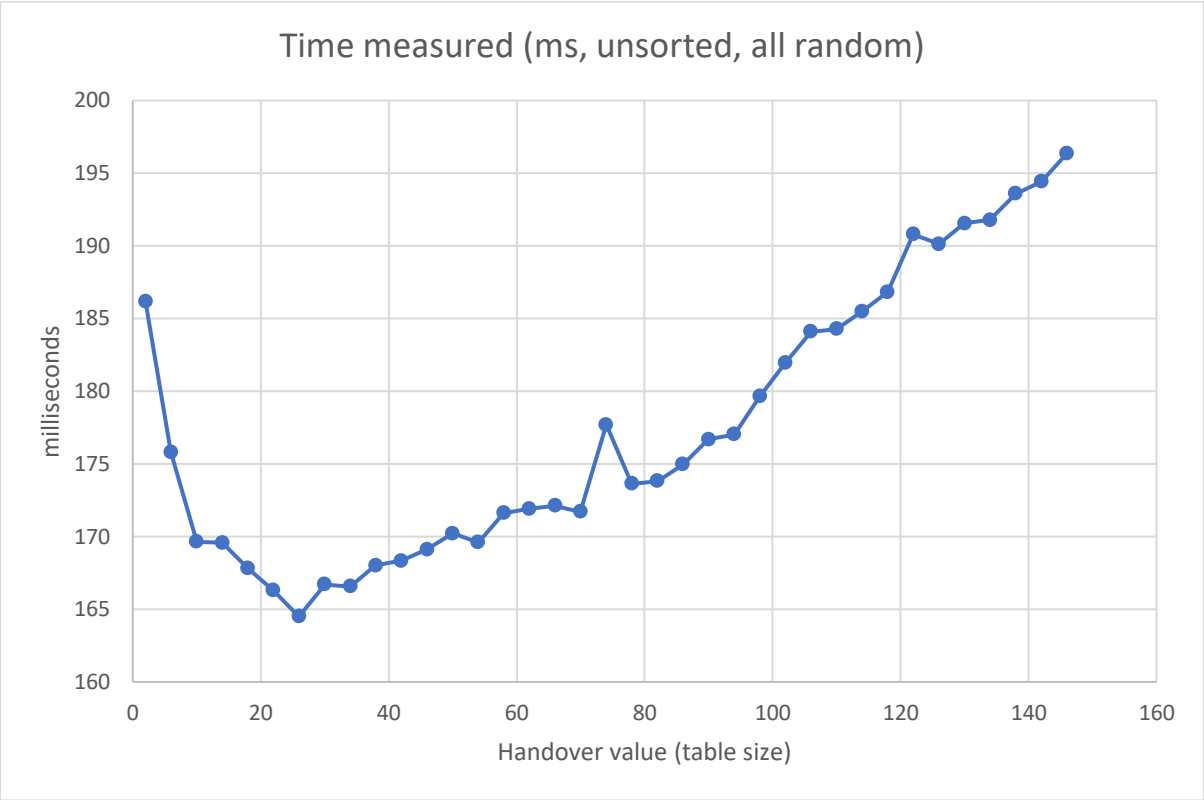
Vi kan ved hjelp av stolpediagrammet se at quicksort med hjelpemetode er vesentlig raskere enn ordinær quicksort. Hvordan dette ble optimalisert (med tanke på deltabellengde), kommer jeg inn på i side 2.

Stolpediagram som sammenligner (med hjelp av fargepar) ordinær quicksort kontra quicksort med innsettingsmetode:




| Sorting method | Time measured (ms) |
|---|---|
| Median3sort, all numbers random | 181,729 |
| Median3sort, all numbers random, sorted | 48,003 |
| Median3sort, every other is random | 126,956 |
| Median3sort, every other is random, sorted | 53,087 |
| Insertion, all numbers random | 163,922 |
| Insertion, all numbers random, sorted | 34,356 |
| Insertion, every other is random | 111,046 |
| Insertion, every other is random, sorted | 42,541 |

Graf som viser tiden målt ved fast tabellstørrelse (2 000 000), men varierende lengde på deltabellene før quicksort gir ansvaret over til innsettingssorteringen (refererer også til dette som «handover-verdi»):




| Handover value | Time measured (ms) |
|---|---|
| 14 | 169,584 |
| 18 | 167,818 |
| 22 | 166,304 |
| 26 | 164,496 |
| 30 | 166,7 |
| 34 | 166,574 |
| 38 | 168,027 |
| 42 | 168,327 |
| 46 | 169,119 |
| 50 | 170,216 |

Vi verifiser sorteringen ved hjelp av to metoder:

1. Sjekker at summen før og etter sorteringen er like.
2. Sjekker at alle elementene i tabellen er større en tallet med én lavere indeks.

```
long sum_before_sort;
long sum_after_sort;
```

```
// Calculate and show the sum of the table before the sort
for (int i = 0; i < 10; i++) {
    printf("%i,", table[i]);
}
sum_before_sort = get_sum(table, table_length);
printf("...\nSum before sort: %li\n\n", sum_before_sort);
```

```
// Calculate and show the sum of the table after the sort
for (int i = 0; i < 10; i++) {
    printf("%i,", table[i]);
}
sum_after_sort = get_sum(table, table_length);
printf("...\nSum after sort: %li\n\n", sum_after_sort);

// Check whether all numbers have been sorted correctly, and if the sum before equals the sum after
int boolean = 1;
for (int i = 1; i < table_length-1; i++) {
    if (table[i] < table[i-1]) {
        printf("The table has NOT been sorted correctly!");
        boolean = 0;
        break;
    }
}
if (boolean == 1) {
    printf("### The table was sorted successfully! ###\n");
}
if (sum_before_sort == sum_after_sort) {
    printf("### The sum before equals the sum after sorting! ###\n\n");
}
```