DAT102 - Obligatorisk innlevering 2

Gruppemedlemmer:
Stephen Neba Fuh, Tord Johan Melheim,
Ebubekir Siddik Yuksel, Casper Eide Özdemir-Børretzen

Uke 6 – Oppgave 1

a)

```java
package src;

public class ParentesSjekker {

    public static boolean sjekk(String s) {
        return sjekk(s, false);
    }

    public static boolean sjekk(String s, boolean debug) {
        if (debug) {
            System.out.println();
            System.out.println(" KJØRER PARENTES SJEKK");
            System.out.println(" - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
            System.out.printf("Streng:%n%s%n%n", s);
        }
        StackADT<Character> par0 = new Stack<>(false);
        // Fyll stabel "par0" med alle parenteser fra strengen
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (c == '(' || c == ')' || c == '[' || c == ']' || c == '{' || c == '}') {
                par0.push(c);
            }
        }
        // Lag ny stabel for åpne parenteser
        StackADT<Character> par1  = new Stack<>(debug);
        while (!par0.isEmpty()) {
            Character c0 = par0.pop();
            Character c1 = par1.peek();
            // Sammenlign parentes på toppen av begge stablene
            if ((c1 != null) && ((c0 == '(' && c1 == ')') || (c0 == '[' && c1 == ']') || (c0 == '{' && c1 == '}'))) {
                // Fjern parentes fra toppen av "åpne parenteser" stabelen hvis match er funnet
                par1.pop();
            }
            else {
                // Push parentesen fra "par0" til "par1" hvis parentesene ikke matcher
                par1.push(c0);
                // Hvis en åpnende parentes er funnet som ikke har lukkende match kan det konkluderes at parentesene
                // i strengen er feilformatert og søket kan avsluttes tidlig
                if (c0 == '(' || c0 == '[' || c0 == '{') {
                    break;
                }
            }
        }
        // Sjekk om det ble funnet match til alle åpne parenteser (par1 stabelen skal i så fall være tom)
        boolean res = par1.isEmpty();
        if (debug) {
            System.out.println("\nParentes sjekk utført: Parenteser er " + (res ? "" : "IKKE ") + "riktig formatert.\n");
        }
        return res;
    }
}
```

b)

```java
package test;

import src.ParentesSjekker;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class ParentesSjekkerTest {
    @Test
    void test() {
        String[] strings = new String[5];
        strings[0] = "{ [ () ] }";
        strings[1] = "{ [ () }";
        strings[2] = "[ ( ) ] }";
        strings[3] = "{ [ ( ] ) }";
        strings[4] = """
            class HelloWorld {
                public static void main(String[] args) {
                System.out.println("Hello World!");
                }
            }""";
        assertTrue(ParentesSjekker.sjekk(strings[0]));
        assertFalse(ParentesSjekker.sjekk(strings[1]));
        assertFalse(ParentesSjekker.sjekk(strings[2]));
        assertFalse(ParentesSjekker.sjekk(strings[3]));
        assertTrue(ParentesSjekker.sjekk(strings[4]));
    }
}
```

Uke 6 – Oppgave 2

a)

```
private static int s(int n) {
    return n == 1 ? 1 : s(n-1) + n;
}

private static int sAntKall(int n) {
    return n == 1 ? 1 : sAntKall(n-1) + 1;
}

public static void main(String[] args) {
    System.out.printf("S_n = S_{n-1} + n, S_1 = 1%nS_{100} = %d%nS_{100} gjennomfører %d kall på s(n) funksjonen.%nn = 100 -> 100
      funksjonskall.%nDette kan uttrykkes som O(n) i O-notasjon.%n", s(100), sAntKall(100));
}

/*
Dette gir resultatet:
    S_n = S_{n-1} + n, S_1 = 1
    S_{100} = 5050
    S_{100} gjennomfører 100 kall på s(n) funksjonen.
    n = 100 -> 100 funksjonskall.
    Dette kan uttrykkes som O(n) i O-notasjon.
*/
```

b)

```
private static boolean isPalindrome(String s) {
    int n = s.length() / 2;
    for (int i = 0; i < n; i++) {
        if (s.charAt(i) != s.charAt(s.length() - 1 - i)) {
            return false;
        }
    }
    return true;
}

public static void main(String[] args) {
    System.out.printf("isPalindrome(\"abba\") = %b%n", isPalindrome("abba"));
    System.out.printf("isPalindrome(\"regninger\") = %b%n", isPalindrome("regninger"));
    System.out.printf("isPalindrome(\"hoyskole\") = %b%n", isPalindrome("hoyskole"));
    System.out.printf("isPalindrome(\"katt\") = %b%n", isPalindrome("katt"));
    System.out.printf("isPalindrome(\"i\") = %b%n", isPalindrome("i"));
}

/*
Dette gir resultatet:
    isPalindrome("abba") = true
    isPalindrome("regninger") = true
    isPalindrome("hoyskole") = false
    isPalindrome("katt") = false
    isPalindrome("i") = true
*/
```

Uke 6 – Oppgave 2

c)

```
private static int fibonacciRek(int n) {
    return n <= 1 ? n : fibonacciRek(n-1) + fibonacciRek(n-2);
}

private static int fibonacciRekAntKall(int n) {
    return 1 + ((n == 0 || n == 1) ? 0 : fibonacciRekAntKall(n-1) + fibonacciRekAntKall(n-2));
}

public static void main(String[] args) {
    for (int n = 0; n <= 20; n++ ) {
        System.out.printf("fibonacciRek(%d) = %d (antall kall: %d)%n", n, fibonacciRek(n), fibonacciRekAntKall(n));
    }
    System.out.printf("%nAntall funksjonskall for fibonacci(n) er lik antall kall for fibonacci(n-1) + antall kall for fibonacci(
     n-2).%nMed andre ord en (nesten) dobbel økning i kjøretid for hver økning av n. I O-notasjon blir dette O(n^2).%n");
}

/*
Dette gir resultatet:
    fibonacciRek(0) = 0 (antall kall: 1)
    fibonacciRek(1) = 1 (antall kall: 1)
    fibonacciRek(2) = 1 (antall kall: 3)
    fibonacciRek(3) = 2 (antall kall: 5)
    fibonacciRek(4) = 3 (antall kall: 9)
    fibonacciRek(5) = 5 (antall kall: 15)
    fibonacciRek(6) = 8 (antall kall: 25)
    fibonacciRek(7) = 13 (antall kall: 41)
    fibonacciRek(8) = 21 (antall kall: 67)
    fibonacciRek(9) = 34 (antall kall: 109)
    fibonacciRek(10) = 55 (antall kall: 177)
    fibonacciRek(11) = 89 (antall kall: 287)
    fibonacciRek(12) = 144 (antall kall: 465)
    fibonacciRek(13) = 233 (antall kall: 753)
    fibonacciRek(14) = 377 (antall kall: 1219)
    fibonacciRek(15) = 610 (antall kall: 1973)
    fibonacciRek(16) = 987 (antall kall: 3193)
    fibonacciRek(17) = 1597 (antall kall: 5167)
    fibonacciRek(18) = 2584 (antall kall: 8361)
    fibonacciRek(19) = 4181 (antall kall: 13529)
    fibonacciRek(20) = 6765 (antall kall: 21891)

    Antall funksjonskall for fibonacci(n) er lik antall kall for fibonacci(n-1) + antall kall for fibonacci(n-2).
    Med andre ord en (nesten) dobbel økning i kjøretid for hver økning av n. I O-notasjon blir dette O(n^2).
*/
```

d)

```
private static int fibonacci(int n) {
    int sumOlder = 0;
    int sumOld = 1;
    int sum = 0;
    for (int i = 0; i <= n; i++) {
        sum = sum + sumOlder;
        sumOlder = sumOld;
        sumOld = sum;
    }
    return sum;
}

public static void main(String[] args) {
    for (int n = 0; n <= 20; n++ ) {
        System.out.printf("fibonacci(%d) = %d (antall interasjoner i løkke: %d)%n", n, fibonacci(n), n);
    }
    System.out.printf("%nAntall ganger løkken kjøres er er n. I O-notasjon blir dette O(n).%n");
}
/*
Dette gir resultatet:
    fibonacci(0) = 0 (antall interasjoner i løkke: 0)
    fibonacci(1) = 1 (antall interasjoner i løkke: 1)
    fibonacci(2) = 1 (antall interasjoner i løkke: 2)
    fibonacci(3) = 2 (antall interasjoner i løkke: 3)
    fibonacci(4) = 3 (antall interasjoner i løkke: 4)
    fibonacci(5) = 5 (antall interasjoner i løkke: 5)
    fibonacci(6) = 8 (antall interasjoner i løkke: 6)
    fibonacci(7) = 13 (antall interasjoner i løkke: 7)
    fibonacci(8) = 21 (antall interasjoner i løkke: 8)
    fibonacci(9) = 34 (antall interasjoner i løkke: 9)
    fibonacci(10) = 55 (antall interasjoner i løkke: 10)
    fibonacci(11) = 89 (antall interasjoner i løkke: 11)
    fibonacci(12) = 144 (antall interasjoner i løkke: 12)
    fibonacci(13) = 233 (antall interasjoner i løkke: 13)
    fibonacci(14) = 377 (antall interasjoner i løkke: 14)
    fibonacci(15) = 610 (antall interasjoner i løkke: 15)
    fibonacci(16) = 987 (antall interasjoner i løkke: 16)
    fibonacci(17) = 1597 (antall interasjoner i løkke: 17)
    fibonacci(18) = 2584 (antall interasjoner i løkke: 18)
    fibonacci(19) = 4181 (antall interasjoner i løkke: 19)
    fibonacci(20) = 6765 (antall interasjoner i løkke: 20)

    Antall ganger løkken kjøres er er n. I O-notasjon blir dette O(n).
*/
```

Uke 6 – Oppgave 3

```
    private static void fig0(int window_width, int window_height, int level, int size, Color color) {
        DrawRectangleLines((window_width - size) / 2, (window_height - size) / 2, size, size, color);
        if (level > 0) {
            fig0(window_width, window_height, level-1, size + 10, color);
        }
    }

    private static void fig1(int x, int y, int level, int size, Color color) {
        DrawRectangleLines(x, y, size, size, color);
        DrawRectangleLines(x, y, size/2, size/2, color);
        DrawRectangleLines(x+size/2, y+size/2, size/2, size/2, color);
        if (level > 0) {
            fig1(x, y, level-1, (int)Math.round(size/2), color);
            fig1(x+size/2, y+size/2, level-1, size/2, color);
        }
    }

    private static void fig2(int x, int y, int level, int size, Color color) {
        DrawTriangleLines(new Vector2().x(x).y(y+size), new Vector2().x(x+size/2).y(y+size), new Vector2().x(x+size/4).y(y+size/2),
         color);
        DrawTriangleLines(new Vector2().x(x+size/2).y(y+size), new Vector2().x(x+size).y(y+size), new Vector2().x(x+size*3/4).y(y+
         size/2), color);
        DrawTriangleLines(new Vector2().x(x+size/4).y(y+size/2), new Vector2().x(x+size*3/4).y(y+size/2), new Vector2().x(x+size/2).y
         (y), color);
        if (level > 0) {
            fig2(x, y+size/2, level-1, size/2, color);
            fig2(x+size/2, y+size/2, level-1, size/2, color);
            fig2(x+size/4, y, level-1, size/2, color);
        }
    }

    public static void main(String[] args) {
        int sel = 0;
        int window_width = 800;
        int window_height = 600;
        String txt = "<- FORRIGE FIGUR / NESTE FIGUR ->";
        SetTraceLogLevel(4);
        InitWindow(window_width, window_height, "Uke 6, Oppgave 3");
        SetTargetFPS(60);
        while (!WindowShouldClose()) {
            BeginDrawing();
            ClearBackground(DARKGRAY);
            if (IsKeyPressed(KEY_LEFT)) { sel = (sel - 1) < 0 ? 2 : (sel - 1); }
            if (IsKeyPressed(KEY_RIGHT)) { sel = (sel + 1) % 3; }
            switch (sel) {
                case 0:
                    fig0(window_width, window_height, 20, 200, LIGHTGRAY);
                    break;
                case 1:
                    fig1((window_width - 400) / 2, (window_height - 400) / 2, 4, 400, LIGHTGRAY);
                    break;
                case 2:
                    fig2((window_width - 400) / 2, (window_height - 400) / 2, 4, 400, LIGHTGRAY);
                    break;
            }
            String fig_txt = "FIGUR " + (sel+1);
            DrawText(fig_txt, (window_width - MeasureText(fig_txt, 20)) / 2, 40, 20, WHITE);
            DrawText(txt, (window_width - MeasureText(txt, 20)) / 2, window_height - 50, 20, WHITE);
            DrawFPS(2, 2);
            EndDrawing();
        }
        CloseWindow();
    }

    /*
    Krever følgende i pom.xml:
      <dependencies>
        <dependency>
          <groupId>uk.co.electronstudio.jaylib</groupId>
          <artifactId>jaylib</artifactId>
          <version>[5.5.0,5.6)</version>
        </dependency>
      </dependencies>
    */
```

Uke 7 – Oppgave 1

```
public class Lfsr {
    private int seed;
    public Lfsr(int seed) { this.seed = seed; }
    public String toString() { return Integer.toString(seed); }
    public int seed() { return seed; }
    public void shift32() {
        seed = ((((seed >> 31) & 1) ^ ((seed >> 21) & 1) ^ ((seed >> 1) & 1) ^ ((seed >> 0) & 1)) == 1) ? ((seed << 1) | 1) : (
     seed << 1);
    }
    public int make32() {
        int result = 0; int mask = 1;
        for (int i = 0; i < 32; i++) {
            shift32();
            if ((seed & 1) == 1) { result |= mask; }
            mask <<= 1;
        }
        return result;
    }
}

public class Sort {
    public static <T extends Comparable<? super T>> void insertion(T[] arr) {
        for (int i = 1; i < arr.length; i++) {
            T tmp = arr[i];
            int j = i;
            while (j > 0 && tmp.compareTo(arr[j-1]) < 0) {
                arr[j] = arr[j---1];
            }
            arr[j] = tmp;
        }
    }

    public static <T extends Comparable<? super T>> void insertionMod(T[] arr) {
        T tmp;
        for (int i = arr.length - 1; i > 0 ; i--) {
            if (arr[i].compareTo(arr[i-1]) < 0) {
                tmp = arr[i];
                arr[i] = arr[i-1];
                arr[i-1] = tmp;
            }
        }
        for (int i = 1; i < arr.length; i += 1) {
            T min = arr[i];
            int j = i - 1;
            if (i + 1 < arr.length - 1) {
                T max = arr[i+1];
                if (max.compareTo(min) < 0) {
                    max = arr[i];
                    min = arr[i+1];
                }
                while (j > 1 && max.compareTo(arr[j]) < 0) {
                    arr[j+2] = arr[j];
                    j -= 1;
                }
                arr[j+2] = max;
                i += 1;
            }
            while (j > 0 && min.compareTo(arr[j]) < 0) {
                arr[j + 1] = arr[j];
                j -= 1;
            }
            arr[j+1] = min;
        }
    }
}

public class Program {
    public static void main(String[] args) {
        int n = 80_000; int s = 1337;
        Lfsr lfsr = new Lfsr(s); Integer[] tab_unsorted = new Integer[n];
        System.out.printf("Lager tabell med %,d tilfeldige heltall (Integer klasse) ved bruk av 32-bit LFSR fra seed-verdi %d.%n"
     , n, s);
        for (int i = 0; i < n; i++) { tab_unsorted[i] = lfsr.make32(); }
        Integer[] tab; long start;

        // Insertion (fra soh1961 github)
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører insertion sort (fra soh1961 github) på tabellen.");
        Sort.insertion(tab);
        System.out.printf(" Insertion sort gjennomført etter %d ms.%n", System.currentTimeMillis() - start);

        // Insertion (modifisert versjon)
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører insertion sort (modifisert versjon) på tabellen.");
        Sort.insertion(tab);
        System.out.printf(" Insertion sort gjennomført etter %d ms.%n", System.currentTimeMillis() - start);
    }
}

//  Dette ga resultatet:
//      Lager tabell med 80,000 tilfeldige heltall (Integer klasse) ved bruk av 32-bit LFSR fra seed-verdi 1337.
//      Kjører insertion sort (fra soh1961 github) på tabellen. Insertion sort gjennomført etter 6781 ms.
//      Kjører insertion sort (modifisert versjon) på tabellen. Insertion sort gjennomført etter 3952 ms.

//  Konklusjon:
//      Da vi testet den opprinnelige algoritmen sammenlignet med den modifiserte,
//      observerte vi at kjøretiden ble betydelig redusert etter endring i algoritmen.
```

Uke 7 – Oppgave 2

a)

```java
public class SortInt {
    private static int tmp;

    // Bubble
    public static void bubble(int[] arr) {
        boolean swapped;
        do {
            swapped = false;
            for (int i = 1; i < arr.length; i++) {
                if (arr[i] < arr[i-1]) {
                    tmp = arr[i];
                    arr[i] = arr[i-1];
                    arr[i-1] = tmp;
                    swapped = true;
                }
            }
        } while (swapped);
    }

    // Selection (utvalgssortering / plukksortering)
    public static void selection(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            int min = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[min]) {
                    min = j;
                }
            }
            if (min != i) {
                tmp = arr[i];
                arr[i] = arr[min];
                arr[min] = tmp;
            }
        }
    }

    // Insertion (Sortering ved innsetting)
    public static void insertion(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int val = arr[i];
            int j = i;
            while (j > 0 && arr[j-1] > val) {
                arr[j] = arr[j-1];
                j -= 1;
            }
            arr[j] = val;
        }
    }

    // Shell (Shellsortering)
    private static final int[][] shell_gaps = new int[][] {
        {701, 301, 132, 57, 23, 10, 4, 1}, // Ciura, 2001
        {1049, 347, 113, 37, 11, 3, 1},    // Rhoads, 2010
    };
    public static void shell(int[] arr) {
        shell(arr, 0);
    }
    public static void shell(int[] arr, int seq) {
        for (int gap : shell_gaps[seq % shell_gaps.length]) {
            for (int i = gap; i < arr.length; i += 1) {
                tmp = arr[i];
                int j = i;
                for (; (j >= gap) && (arr[j - gap] > tmp); j -= gap) {
                    arr[j] = arr[j - gap];
                }
                arr[j] = tmp;
            }
        }
    }

    // Merge (Flettesortering)
    public static void merge(int[] arr) {
        int[] aux = Arrays.copyOf(arr, arr.length);
        merge(arr, aux, 0, arr.length - 1);
    }

    private static void merge(int[] arr, int[] aux, int lo, int hi) {
        if (hi <= lo) { return; }
        int mid = lo + ((hi - lo) / 2);
        merge(arr, aux, lo, mid);
        merge(arr, aux, mid+1, hi);
        int i = lo;
        int k = lo;
        for (int j = mid+1; i <= mid && j <= hi; k++) { aux[k] = (arr[i] <= arr[j]) ? arr[i++] : arr[j++]; }
        while(i <= mid) { aux[k++] = arr[i++]; }
        for (i = lo; i <= hi; i++) { arr[i] = aux[i]; }
    }
```

Uke 7 – Oppgave 2

a)

```java
    // Heap (Haugsortering (forbedret versjon av utvalgssortering))
    public static void heap (int[] arr) {
        for (int i = (arr.length - 2) / 2; i >= 0; i--) { heapify(arr, i, arr.length); }
        for (int i = arr.length; i > 0; i--) {
            tmp = arr[0];
            arr[0] = arr[i-1];
            arr[i-1] = tmp;
            heapify(arr, 0, i-1);
        }
    }

    private static void heapify(int[] arr, int i, int size) {
        int left = 2 * i + 1;
        int right = 2 * i + 2;
        int max = (left < size && arr[left] > arr[i]) ? left : i;
        if (right < size && arr[right] > arr[max]) { max = right; }
        if (max != i) {
            tmp = arr[i];
            arr[i] = arr[max];
            arr[max] = tmp;
            heapify(arr, max, size);
        }
    }

    // Quick, Hoare partition scheme (Kvikksortering)
    public static void quick_hoare(int[] arr) {
        if (arr.length > 1) { quick_hoare(arr, 0, arr.length - 1); }
    }

    private static void quick_hoare(int[] arr, int lo, int hi) {
        if (lo < hi) {
            int i = lo - 1;
            int j = hi + 1;
            while (true) {
                do { i += 1; } while (arr[i] < arr[lo]); // Pivot -> arr[lo]
                do { j -= 1; } while (arr[j] > arr[lo]); // Pivot -> arr[lo]
                if (i >= j) { break; }
                tmp = arr[i];
                arr[i] = arr[j];
                arr[j] = tmp;
            }
            quick_hoare(arr, lo, j);
            quick_hoare(arr, j+1, hi);
        }
    }

    // Quick, Dutch national flag (Kvikksortering)
    public static void quick_dutch_flag(int[] arr) {
        if (arr.length > 1) { quick_dutch_flag(arr, 0, arr.length - 1); }
    }

    private static void quick_dutch_flag(int[] arr, int lo, int hi) {
        if (lo < hi) {
            int pivot = arr[hi];
            int i = lo;
            int j = lo;
            int k = hi;
            while (j <= k) {
                if (arr[j] < arr[hi]) { // Pivot -> arr[hi]
                    tmp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = tmp;
                    i += 1;
                    j += 1;
                }
                else if (arr[j] > arr[hi]) { // Pivot -> arr[hi]
                    tmp = arr[k];
                    arr[k] = arr[j];
                    arr[j] = tmp;
                    k -= 1;
                }
                else { j += 1; }
            }
            quick_dutch_flag(arr, lo, i-1);
            quick_dutch_flag(arr, j, hi);
        }
    }

    // Counting
    public static void counting(int[] arr) {
        Map<Integer, Integer> freq = new TreeMap<>();
        for (int n : arr) { freq.put(n, freq.getOrDefault(n, 0) + 1); }
        int i = 0;
        for (Map.Entry<Integer, Integer> entry : freq.entrySet()) {
            int key = entry.getKey();
            for (int val = entry.getValue(); val > 0; val--) { arr[i++] = key; }
        }
    }
}
```

Uke 7 – Oppgave 2

a)

```
public class Program {
    private static void sort(int n) {
        int[] tab_unsorted = new int[n];
        int s = 1312;
        Lfsr lfsr = new Lfsr(s);
        System.out.printf("\nLager tabell med %,d tilfeldige heltall ved bruk av 32-bit LFSR fra seed-verdi %d.\n", n, s);
        for (int i = 0; i < n; i++) { tab_unsorted[i] = lfsr.make32(); }
        int[] tab;
        long start;

        // Bubble
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører bubble sort på tabellen.");
        SortInt.bubble(tab);
        System.out.printf(" Bubble sort gjennomført etter %d ms.\n", System.currentTimeMillis() - start);

        // Selection
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører selection sort på tabellen.");
        SortInt.selection(tab);
        System.out.printf(" Selection sort gjennomført etter %d ms.\n", System.currentTimeMillis() - start);

        // Insertion
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører insertion sort på tabellen.");
        SortInt.insertion(tab);
        System.out.printf(" Insertion sort gjennomført etter %d ms.\n", System.currentTimeMillis() - start);

        // Merge
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører merge sort på tabellen.");
        SortInt.merge(tab);
        System.out.printf(" Merge sort gjennomført etter %d ms.\n", System.currentTimeMillis() - start);

        // Heap
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører heapsort på tabellen.");
        SortInt.heap(tab);
        System.out.printf(" Heapsort gjennomført etter %d ms.\n", System.currentTimeMillis() - start);

        // Quick
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører quicksort (Hoare) på tabellen.");
        SortInt.quick_hoare(tab);
        System.out.printf(" Quicksort gjennomført etter %d ms.\n", System.currentTimeMillis() - start);

        // Quick
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører quicksort (Dutch national flag) på tabellen.");
        SortInt.quick_dutch_flag(tab);
        System.out.printf(" Quicksort gjennomført etter %d ms.\n", System.currentTimeMillis() - start);

        // Shell
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører shellsort (Ciura) på tabellen.");
        SortInt.shell(tab);
        System.out.printf(" Shellsort gjennomført etter %d ms.\n", System.currentTimeMillis() - start);

        // Shell
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører shellsort (Rhoads) på tabellen.");
        SortInt.shell(tab, 1);
        System.out.printf(" Shellsort gjennomført etter %d ms.\n", System.currentTimeMillis() - start);

        // Counting
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører counting sort på tabellen.");
        SortInt.counting(tab);
        System.out.printf(" Counting sort gjennomført etter %d ms.\n", System.currentTimeMillis() - start);
    }

    public static void main(String[] args) {
        sort(32_000);
        sort(64_000);
        sort(128_000);
    }
}
```

Uke 7 – Oppgave 2

a) Resultater:

Målt med tabell med n antall unike 32-bit int verdier.

Bubble sort (best: n / avg: n^2 / worst: n^2):

Avg (n^2) brukt for teoretisk tid.

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $\big(c \cdot f(n)\big)$
32.000	10	1914 ms	$1.87 \cdot 10^{-6} \cdot n^2$
64.000	10	6260 ms	$1.53 \cdot 10^{-6} \cdot n^2$
128.000	10	25907 ms	$1.58 \cdot 10^{-6} \cdot n^2$

Selection sort (best: n^2 / avg: n^2 / worst: n^2):

Avg (n^2) brukt for teoretisk tid.

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $\big(c \cdot f(n)\big)$
32.000	10	270 ms	$2.63 \cdot 10^{-7} \cdot n^2$
64.000	10	852 ms	$2.08 \cdot 10^{-7} \cdot n^2$
128.000	10	3422 ms	$2.09 \cdot 10^{-7} \cdot n^2$

Insertion sort (best: n / avg: n^2 / worst: n^2):

Avg (n^2) brukt for teoretisk tid.

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $\big(c \cdot f(n)\big)$
32.000	10	70 ms	$6.84 \cdot 10^{-8} \cdot n^2$
64.000	10	269 ms	$6.57 \cdot 10^{-8} \cdot n^2$
128.000	10	1102 ms	$6.72 \cdot 10^{-8} \cdot n^2$

Counting sort (avg: $n + r$ / worst: $n + r$):

Avg ($n + r$) brukt for teoretisk tid.
Alle tilfeldige tall i den unsorterte tabellen var unike, så $r = n$.

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $\big(c \cdot f(n)\big)$
32.000	10	12 ms	$1.88 \cdot 10^{-4} \cdot (n + r)$
64.000	10	24 ms	$1.88 \cdot 10^{-4} \cdot (n + r)$
128.000	10	54 ms	$2.10 \cdot 10^{-4} \cdot (n + r)$
1.024.000	10	751 ms	$3.67 \cdot 10^{-4} \cdot (n + r)$

Shellsort (Ciura) (best $n \log n$ / worst: $n \log^2 n$):

Worst ($n \log^2 n$) brukt for teoretisk tid.

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $\big(c \cdot f(n)\big)$
32.000	10	3 ms	$4.19 \cdot 10^{-7} \cdot n \log^2 n$
64.000	10	6 ms	$3.68 \cdot 10^{-7} \cdot n \log^2 n$
128.000	10	15 ms	$4.07 \cdot 10^{-7} \cdot n \log^2 n$
1.024.000	10	443 ms	$1.09 \cdot 10^{-6} \cdot n \log^2 n$

Shellsort (Rhoads) (best $n \log n$ / worst: $n \log^2 n$):

Worst ($n \log^2 n$) brukt for teoretisk tid.

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $\big(c \cdot f(n)\big)$
32.000	10	2 ms	$2.79 \cdot 10^{-7} \cdot n \log^2 n$
64.000	10	5 ms	$3.06 \cdot 10^{-7} \cdot n \log^2 n$
128.000	10	13 ms	$3.53 \cdot 10^{-7} \cdot n \log^2 n$
1.024.000	10	341 ms	$8.35 \cdot 10^{-7} \cdot n \log^2 n$

Uke 7 – Oppgave 2

a) **Heapsort (best:** $n \log n$ **/ avg:** $n \log n$ **/ worst:** $n \log n$**):**

Avg $(n \log n)$ brukt for teoretisk tid.

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $\left(c \cdot f(n)\right)$
32.000	10	2 ms	$4.18 \cdot 10^{-6} \cdot n \log n$
64.000	10	5 ms	$4.89 \cdot 10^{-6} \cdot n \log n$
128.000	10	12 ms	$5.53 \cdot 10^{-6} \cdot n \log n$
1.024.000	10	119 ms	$5.82 \cdot 10^{-6} \cdot n \log n$
16.384.000	10	3302 ms	$8.41 \cdot 10^{-6} \cdot n \log n$

Merge sort (best: $n \log n$ **/ avg:** $n \log n$ **/ worst:** $n \log n$**):**

Avg $(n \log n)$ brukt for teoretisk tid.

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $\left(c \cdot f(n)\right)$
32.000	10	4 ms	$8.35 \cdot 10^{-6} \cdot n \log n$
64.000	10	5 ms	$4.89 \cdot 10^{-6} \cdot n \log n$
128.000	10	10 ms	$4.60 \cdot 10^{-6} \cdot n \log n$
1.024.000	10	104 ms	$5.09 \cdot 10^{-6} \cdot n \log n$
16.384.000	10	1973 ms	$5.02 \cdot 10^{-6} \cdot n \log n$

Quicksort (Dutch national flag) (best: $n \log n$ **/ avg:** $n \log n$ **/ worst:** n^2**):**

Avg $(n \log n)$ brukt for teoretisk tid.

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $\left(c \cdot f(n)\right)$
32.000	10	2 ms	$4.18 \cdot 10^{-6} \cdot n \log n$
64.000	10	4 ms	$3.91 \cdot 10^{-6} \cdot n \log n$
128.000	10	9 ms	$4.14 \cdot 10^{-6} \cdot n \log n$
1.024.000	10	90 ms	$4.40 \cdot 10^{-6} \cdot n \log n$
16.384.000	10	1718 ms	$4.38 \cdot 10^{-6} \cdot n \log n$

Quicksort (Hoare) (best: $n \log n$ **/ avg:** $n \log n$ **/ worst:** n^2**):**

Avg $(n \log n)$ brukt for teoretisk tid.

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $\left(c \cdot f(n)\right)$
32.000	10	2 ms	$4.18 \cdot 10^{-6} \cdot n \log n$
64.000	10	5 ms	$4.89 \cdot 10^{-6} \cdot n \log n$
128.000	10	9 ms	$4.14 \cdot 10^{-6} \cdot n \log n$
1.024.000	10	85 ms	$4.16 \cdot 10^{-6} \cdot n \log n$
16.384.000	10	1559 ms	$3.97 \cdot 10^{-6} \cdot n \log n$

Uke 7 – Oppgave 2

b) Vi legger en ny type kvikksortering som bruker Lomuto metoden til i SortInt klassen:

```
public class SortInt {

    // ...

    // Quick, Lomuto partition scheme (Kvikksortering) - Best: n log n / Avg: n log n / Worst: n^2
    public static void quick_lomoto(int[] arr) {
        if (arr.length > 1) { quick_lomoto(arr, 0, arr.length - 1); }
    }

    private static void quick_lomoto(int[] arr, int lo, int hi) {
        if (lo < hi) {
            int p = lo; // Pivot index
            for (int i = lo; i < hi; i++) {
                if (arr[i] <= arr[hi]) { // Pivot -> arr[hi]
                    tmp = arr[i];
                    arr[i] = arr[p];
                    arr[p] = tmp;
                    p += 1;
                }
            }
            tmp = arr[hi];
            arr[hi] = arr[p];
            arr[p] = tmp;
            quick_lomoto(arr, lo, p-1);
            quick_lomoto(arr, p+1, hi);
        }
    }

    // ...
}
```

Så kjører vi tester med en tabell der alle elementer er like:

```
public class Program {
    private static void sort(int n) {
        int[] tab_unsorted = new int[n];
        System.out.printf("%nLager tabell med %,d heltall satt til den samme verdien.%n", n);
        for (int i = 0; i < n; i++) { tab_unsorted[i] = 99; }
        int[] tab;
        long start;

        // Quick (Lomuto) - Best: n log n / Avg: n log n / Worst: n^2
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører quicksort (Lomuto) på tabellen.");
        SortInt.quick_lomoto(tab);
        System.out.printf(" Quicksort gjennomført etter %d ms.%n", System.currentTimeMillis() - start);

        // Quick (Hoare) - Best: n log n / Avg: n log n / Worst: n^2
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører quicksort (Hoare) på tabellen.");
        SortInt.quick_hoare(tab);
        System.out.printf(" Quicksort gjennomført etter %d ms.%n", System.currentTimeMillis() - start);

        // Quick (Dutch national flag) - Best: n log n / Avg: n log n / Worst: n^2
        tab = Arrays.copyOf(tab_unsorted, n);
        start = System.currentTimeMillis();
        System.out.printf("Kjører quicksort (Dutch national flag) på tabellen.");
        SortInt.quick_dutch_flag(tab);
        System.out.printf(" Quicksort gjennomført etter %d ms.%n", System.currentTimeMillis() - start);
    }

public static void main(String[] args) {
        sort(32_000);
        sort(64_000);
        sort(128_000);
    }
}
```

Uke 7 – Oppgave 2

b) Resultater:

Målt med tabell med n antall like 32-bit int verdier.

Quicksort (Lomuto) (best: $n \log n$ / avg: $n \log n$ / worst: n^2):

Alle elmenter i tabellen like.

N	Antall målinger	Målt tid (gjennomsnitt)
16.000	10	51 ms
32.000	10	ERR! Stack overflow!

Quicksort (Hoare) (best: $n \log n$ / avg: $n \log n$ / worst: n^2):

Alle elmenter i tabellen like.

N	Antall målinger	Målt tid (gjennomsnitt)
16.000	10	1 ms
32.000	10	1 ms
64.000	10	2 ms
128.000	10	2 ms
1.024.000	10	20 ms
16.384.000	10	275 ms

Quicksort (Dutch national flag) (best: $n \log n$ / avg: $n \log n$ / worst: n^2):

Alle elmenter i tabellen like.

N	Antall målinger	Målt tid (gjennomsnitt)
16.000	10	1 ms
32.000	10	1 ms
64.000	10	1 ms
128.000	10	1 ms
1.024.000	10	1 ms
16.384.000	10	10 ms

Konklusjon:

Kvikksorteringsalgoritmen som tok i bruk Lomutos metode ga dårlig ytelse når alle elementene i tabellen var like. I tillegg til at partisjoneringsprosessen førte til høyt antall rekursive kall som ga "Stack Overflow" error ved $n \gtrapprox 20000$. Hoares metode ga bedre ytelse og færre rekursive kall. Til slutt testet vi "nederlandske flaggets problem"-metoden, som er spesiellt egnet for sortering av repeterende elementer, og ga som forventet laveste kjøretid i våre tester.