Lab 2 – Sorting

1. **Explain the code from assignment 1, use the input [1, 2, 4, 3, 5, 0]**
   På den första uppgiften använder vi Insertionsort för att sortera en lista med ints (heltal). Användaren skriver in hur många heltal som ska sorteras, och sedan matas siffrorna in via terminalen (stdin). Den första metoden (InsertionSort) anropas och då skickas en array med sifforna vi skickade in. Siffrorna sorteras med sorteringsalgoritmen och varje steg syns sedan i terminalen.

   Tidskomplexitet: O(n^2)
   Stabil och in place
   Bra för: Små arrayer, när minne spelar roll, nästan sorterad array

2. **Explain the code from assignment 2, use the input [1, 2, 4, 3, 5, 0]**
   I InsertionSort metoden finns ett metodsanrop till metoden exch. I den metoden så byter element som ska sorteras plats med varandra. Det finns även en counter i exch metoden som räknas antalet swaps, så att dessa kan skrivas ut och visas på terminalen för användaren.

3. **Explain the code from assignment 3 (inversion count) and how you calculate its time complexity, use the input [1, 2, 4, 3, 5, 0]**
   Metoden inversions kollar på alla inversioner I en array och skriver sedan ut ALLA inversionen så att de syns. En counter finns i slutet av metoden och räknar då alla inversioner som sker.

   Tidskomplexitet: Eftersom det finns 2 for loopar i metoden → O(n^2)

4. **Explain the code from assignment 4 (separating negatives from positives) and how you calculate its time complexity.**
   Metoden får in en array med siffror och storleken av arrayen. Vi skapar en variabel j och temp. I for loopen är villkoret, om det som är på a[i] platsen är mindre än noll, alltså negativt, så kommer stegen i if satsen att ske.

   Tidskomplexitet: I värsta fall blir tiden Linjär O(n), eftersom vi i värsta fall måste gå igenom hela arrayen om den negativa siffran är i slutet av arrayen. For loopen måste loopa genom hela för att hitta den.
   Minne: Vi har konstant minne O(1) eftersom vi inte skapar några nya minnesplatser.

5. **Explain the code from assignment 5 and how you performed the time measurements, why your results are valid and show the results.**

I denna uppgift jämfördes tiden för att sortera olika storleker av arrayer med sorteringsalgoritmerna MergeSort och InsertionSort. För att mäta tiden i koden användes den inbyggda metoden nanotime(). Metoden returnerar en precis tid i nanosekunder. Vi klockar tiden innan arrayen sorteras, och sedan klockar tiden efter arrayen sorteras. Sedan tar vi skillnaden i tiden och konverterar tiden från nano till sekunder. Vi gör samma sak för Insertionsort för att se hur lång tid det tar att sortera den.

RESULTAT:

ArraySize	InsertionSort (sec)	MergeSort (sec)
a [10]	2.6036E-5	0.042231471
a [100]	5.91367E-4	0.042878226
a [1000]	0.056309547	0.05197164
a [10000]	0.263474203	0.1793005
a [100000]	10.745480562	4.160788106
a [1000000]	-	-

Vi kan se att InsertionSort är snabbare för mindre arrayer än Mergesort, medan MergeSort är snabbare för stora arrayer.

MergeSort: Tidskomplexitet: O(NlogN)

InsertionSort: Tidskomplexitet: O(N^2)