# 1. Beskrivning av Urvalssortering (eng. Selection sort)

Urvalssortering är en av de enklaste sorteringsalgoritmerna. Den delar en lista i två delar, en sorterad del och en osorterad (från början finns det ingen sorterad del).

- Algoritmen letar upp det minsta talet i listan och lägger det först samtidigt som det första talet läggs där det minsta låg. Nu ligger det minsta först. Vi har en sorterad del (1 tal) och en osorterad del.
- Sedan går algoritmen igenom den osorterade delen av listan och letar efter det minsta talet och när den hittar det så placeras det i andra positionen; position 1 och 2 är nu sorterade.
- Sedan går den igenom resten av listan och letar efter det minsta talet och när den hittar det så placeras det i tredje positionen; position 1, 2 och 3 är nu sorterade.

Man ser nu att processen upprepas. Programmet ska alltid gå igenom en lista och söka upp det minsta talet (lämpligen en def). Den del av listan som ska genomsökas blir kortare och kortare men det är samma procedur. Sedan ska det nyfunna minsta talet bytas med första osorterade talet. Och sedan startar det om igen.

Vi kommer behöva nästlade loopar. En loop för att hålla reda på, steg för steg, var gränsen är mellan sorterade och osorterade tal, ett index. Sedan behöver vi en loop för att gå igenom talen i den osorterade delen och leta efter det minsta.

Vi tar oss igenom ett kort exempel. Kalla listan L, den går från index 0 till index 7, 8 st tal ska sorteras.

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| tal | 36 | 22 | 10 | 23 | 34 | 78 | 9 | 40 |

I starten sätter vi minst=36, minst_i=0. Första talet i det som kommer att bli den sorterade listan. Vi inför också osort_start=0.

Sedan låter vi j=1. Första talet i osorterade listan. Eftersom L[1] är mindre än minst (som är 36) sätter vi minst=22 och minst_i=1.

Sedan j=2. L[2] är mindre än minst. L[2]=10 och minst=22. Vi sätter minst=10 och minst_i=2.

Sedan j=3. L[3], dvs 23, är inte mindre än minst. Vi går vidare. (minst=10, minst_i=2)

Sedan j=4. L[4], dvs 34, är inte mindre än minst. Vi går vidare. (minst=10, minst_i=2)

Sedan j=5. L[5], dvs 78, är inte mindre än minst. Vi går vidare. (minst=10, minst_i=2)

Sedan j=6. L[6], dvs 9, är mindre än minst. Vi sätter minst=9 och minst_i=6.

Sedan j=7. L[7], dvs 40, är inte mindre än minst. Vi är klara och vet att det minsta talet är 9 och finns på plats med index_i=6. Vi byter nu plats på 36 och 9 och får listan:

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| tal | 9 | 22 | 10 | 23 | 34 | 78 | 36 | 40 |

Vi har nu verkligen det minsta talet först.
Nu upprepas algoritmen men vi startar inte med L[0] utan L[1]. Detta håller osort_start loopen reda på. L[0] är den sorterade delen och L[1] till L[7] är osorterad. Så när vi upprepar startar vi med minst=22 och minst_i=1, j=2. Sedan stegar vi igenom j=3, j=4 osv till j=7. osort_start är den yttre loopen och j är den inre loopen.