Algoritmi

L'esonero di informatica parte teoria ha una struttura così definita:

- **Esercizio 1:** algoritmo di ordinamento (bubble, insertion e selection) o ricerca (binary)
- **Esercizio 2:** disegnare un flow chart di un algoritmo.
- **Esercizio 3:** esercizio in C.

Binary Search

Precondizioni: l'array deve essere ordinato con una dimensione.

Input: un numero x, un array e la dimensione.

Complessità: $O(log_2(DIM))$.

Invariante: l'invariante è la variabile x che occorre in arr da posizione inizio a posizione fine.

Spiegazione: questo algoritmo si basa sul controllare l'array dividendolo a metà ogni volta.

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 8

int main(int argc, char const *argv[])
{
    int array[DIM] = {1, 3, 5, 7, 9, 12, 16, 20};
    int inizio = 0, fine = DIM - 1;
  int medio = (inizio + fine) / 2;
  int posizione = -1;
```

```
  while((inizio <= fine) && (posizione == -1))
  {
      if(array[medio] == x)
      {
          posizione = medio;
      }
      else
      {
          if(array[medio] > x)
          {
              fine = medio - 1;
          }
          else
          {
              inizio = medio + 1;
          }
          medio = (inizio + fine) / 2;
      }
  }
  printf("Posizione: %d\n", posizione);
    return 0;
}
```

Insertion Sort

Precondizioni: nessuno.

Input: un array e la dimensione.

Spiegazione: questo algoritmo si basa sul analizzare un elemento e se l'array che si forma a ogni ciclo dietro a lui è più grande posiziona quell'elemento in quella posizione.

Complessità: $O(DIM^2)$.

Invariante: l'invariante è l'array ordinato dalla posizione 0 alla posizione i.

```c
#include <stdio.h>
#include <stdlib.h>
#define DIM 8

int main(int argc, char const *argv[])
{
    int array[DIM] = {1, 3, 5, 7, 9, 12, 16, 20};
    int i, j, tmp;
  for(i = 1; i < DIM; i++)
  {
        tmp = array[i];
        j = i - 1;
        /* Serve a spostare il vettore avanti */
        while(j >= 0 && array[j] > tmp)
        {
            array[j + 1] = array[j];
            j--;
        }
        /* Riempie il vuoto dello shift */
        array[j + 1] = tmp;
  }
    return 0;
}
```

Selection Sort

Precondizioni: nessuno.

Input: un array e la dimensione.

Complessità: $O(DIM^2)$.

Invariante: l'invariante è l'array ordinato dalla posizione 0 alla posizione i.

Spiegazione: bisogna trovare il numero più piccolo e scambiarlo con il successivo, ogni volta che si posiziona un numero all'inizio non bisogna più controllarlo.

```c
#include <stdio.h>
#include <stdlib.h>
#define DIM 8

int main(int argc, char const *argv[])
{
    int array[DIM] = {1, 3, 5, 7, 9, 12, 16, 20};
    int i, j, min_index, tmp;
  for(i = 0; i < DIM; i++)
  {
        min_index = i;
        for(j = i; j < DIM; j++)
        {
            if(array[j] < array[min_index])
            {
                min_index = j;
            }
        }
        tmp = array[min_index];
        array[min_index] = array[i];
        array[i] = tmp;
  }
    return 0;
}
```

Bubble Sort

Precondizioni: nessuno.

Input: un array e la dimensione.

Complessità: $O(DIM^2)$.

Invariante: l'invariante è l'array ordinato dalla posizione 0 alla posizione i.

Spiegazione: questo algoritmo serve a spostare l'elemento più grande in questione verso la fine e poi bloccarlo lì per non sprecare calcoli. Il controllo

avviene in coppia e se il primo è maggiore si scambia di posizione.

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 8

int main(int argc, char const *argv[])
{
    int array[DIM] = {1, 3, 5, 7, 9, 12, 16, 20};
    int i, j, tmp;
  for(i = (DIM - 1); i > 0; i--)
  {
      for(j = 0; j < i; j++)
      {
          if(array[j] > array[j + 1])
          {
              tmp = array[j];
              array[j] = array[j + 1];
              array[j + 1] = tmp;
          }
      }
  }
    return 0;
}
```