

# Elaborato Assembly

Corso di Architettura degli Elaboratori A.A. 2023/2024  
Prof. Franco Fummi, Prof. Michele Lora

Tommi Bimbato VR500751, Antonio Iovine VR504083

18 giugno 2024

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Approccio progettuale	1
1.2	Analisi delle specifiche	1
1.2.1	Input	1
1.2.2	Algoritmi di pianificazione	2
1.3	Sviluppo	3
1.3.1	Acquisizione e struttura dati	4
1.3.2	Ordinamento dei prodotti	4
1.3.3	Output	4
1.3.4	Penalità	4
1.4	Codice	5
1.4.1	Struttura della cartella di progetto	5
1.4.2	File sorgenti	5
1.5	Scelte progettuali	12
1.5.1	Output su file / funzioni extra-specifiche	12
1.5.2	Algoritmi di sorting	12
1.5.3	Limiti del software	12

## **Sommario**

L'obiettivo del progetto è sviluppare un software per la pianificazione delle attività di un sistema produttivo. La produzione è organizzata in slot temporali uniformi, durante i quali un solo prodotto può essere in fase di lavorazione. Il software consentirà di ottimizzare la pianificazione delle attività secondo due algoritmi di pianificazioni differenti. L'intero software è stato sviluppato in linguaggio Assembly (Sintassi AT&T) e testato su un insieme di dati di prova allegati a questa documentazione.

## Capitolo 1

# Introduzione

### 1.1 Approccio progettuale

L'elaborato è stato condotto seguendo un approccio metodologico strutturato. Inizialmente, è stata eseguita un'analisi dettagliata per identificare i requisiti e le funzionalità principali del software. Questo processo ha consentito una comprensione completa del contesto operativo e degli obiettivi da raggiungere.

Successivamente, è stata sviluppata una bozza del software utilizzando il linguaggio di programmazione C. Questo ha permesso la traduzione dei requisiti in una struttura logica e l'identificazione dell'architettura generale del software.

Parallelamente, sono stati definiti gli spazi di memoria necessari per la memorizzazione dei dati durante l'esecuzione del programma. Ciò ha garantito un utilizzo efficiente delle risorse disponibili.

Infine, sono stati eseguiti test approfonditi per verificare il corretto funzionamento del programma e identificare eventuali aree di miglioramento. L'iterazione su questo processo ha portato a modifiche e ottimizzazioni fino al raggiungimento di un livello soddisfacente di prestazioni e funzionalità.

### 1.2 Analisi delle specifiche

La produzione è organizzata in slot temporali uniformi, durante i quali un solo prodotto può essere in fase di lavorazione. Il programma analizza una serie di prodotti, ognuno caratterizzato da un identificativo, una durata, una scadenza e una priorità secondo le specifiche seguenti:

- Identificativo: un codice da 1 a 127;
- Durata: il numero di slot temporali per il completamento (da 1 a 10);
- Scadenza: il limite massimo di tempo entro cui il prodotto deve essere completato (da 1 a 100);
- Priorità: un valore da 1 a 5, che indica sia la priorità che la penalità per il ritardo sulla scadenza<sup>1</sup>.

Al termine della pianificazione, il programma calcolerà la penale dovuta agli eventuali ritardi di produzione.

#### 1.2.1 Input

L'utente invoca il programma "pianificatore" e fornisce due file come parametri da linea di comando, il primo viene considerato input, mentre il secondo viene utilizzato per salvare i risultati della pianificazione. Ad esempio:q

```
./Pianificatore Ordini.txt output.txt
```

---

<sup>1</sup>Il valore 5 indica la priorità più alta.

In questo caso, il programma caricherà gli ordini dal file `Ordini.txt` e salverà le statistiche stampate a video nel file `output.txt`. Se l'utente fornisce solo un parametro, il salvataggio della pianificazione su file verrà ignorato.

Il file degli ordini dovrà avere un prodotto per riga, con tutti i parametri separati da virgola. Ad esempio, se l'ordine fosse:

```
ID: 4; Durata: 10; Scadenza: 12; Priorità: 4;
```

Il file dovrà contenere la seguente riga:

```
4,10,12,4
```

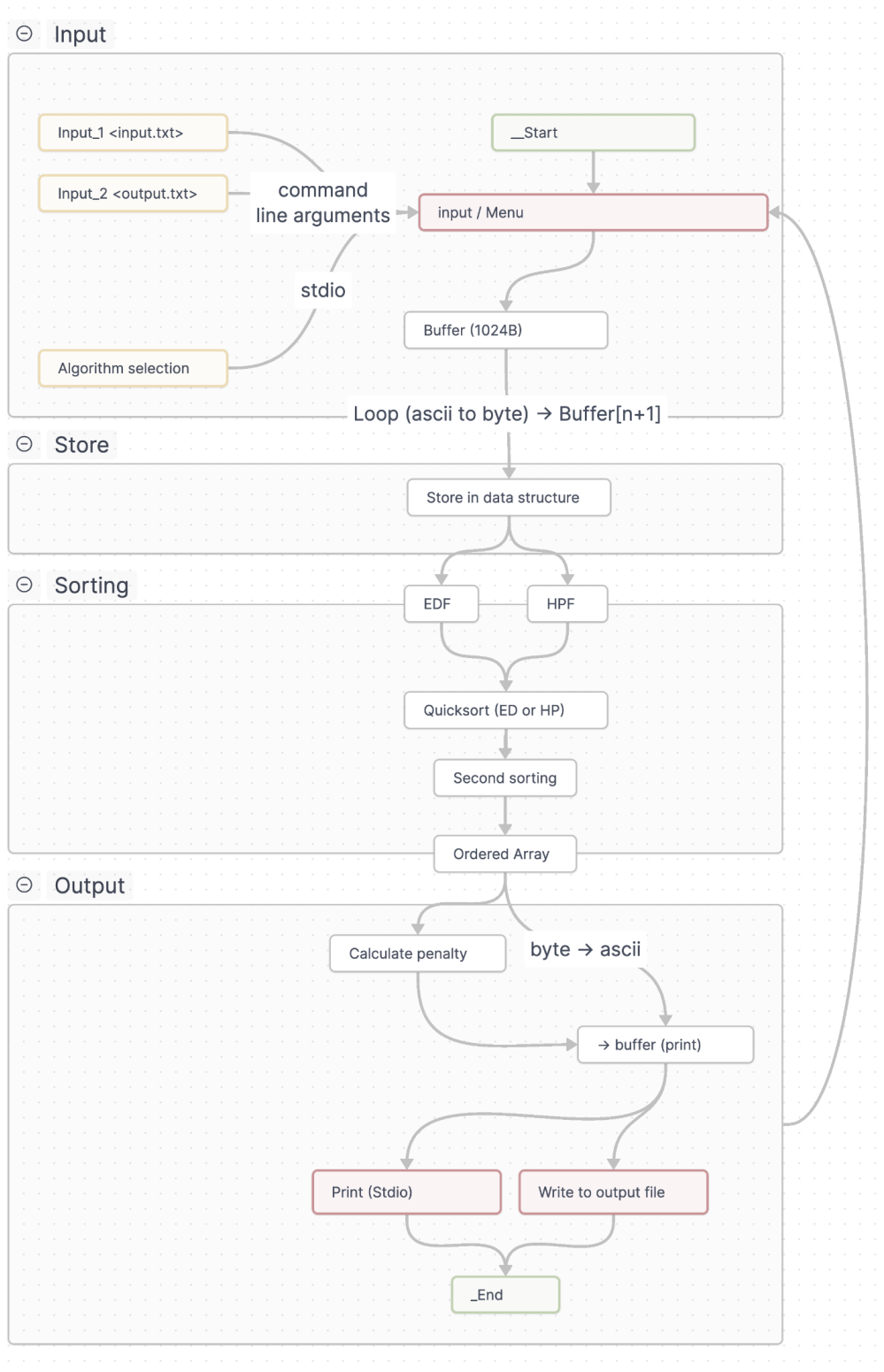
### 1.2.2 Algoritmi di pianificazione

Una volta letto il file, il programma visualizzerà il menu principale, permettendo all'utente di selezionare l'algoritmo di pianificazione desiderato. Le opzioni disponibili sono:

1. Earliest Deadline First (EDF): Si pianificano per primi i prodotti con scadenza più vicina. In caso di parità nella scadenza, si considera la priorità più alta.
2. Highest Priority First (HPF): Si pianificano per primi i prodotti con la priorità più alta. In caso di parità di priorità, si considera la scadenza più vicina.

L'utente può selezionare uno dei due algoritmi per la pianificazione delle attività del sistema produttivo.

## 1.3 Sviluppo



**Figura 1.1:** Schema base del funzionamento del software. [alcune funzionalità sono state omesse per leggibilità dello schema.]

### 1.3.1 Acquisizione e struttura dati

Il software acquisisce l'intero contenuto del file di input e lo memorizza in una variabile nel buffer, parallelamente ne calcola la lunghezza (numero di righe) per allocare la memoria necessaria per la struttura dati vera e propria. Successivamente, il programma procede a leggere il contenuto del buffer, converte le informazioni dal formato ASCII al formato numerico e le memorizza in un array di struct. La peculiarità di questa struttura dati è che lo spazio occupato da un prodotto (ID, durata, scadenza e priorità) è di 4 byte totali, pari alla dimensione di un registro interno del processore considerato per l'applicazione del software.

### 1.3.2 Ordinamento dei prodotti

Acquisita la struttura dati, il programma acquisisce la scelta dell'algoritmo di pianificazione da parte dell'utente e procede con l'ordinamento dei prodotti in base alla scelta effettuata. Il sorting è gestito da una variabile che definisce la priorità di ordinamento (HPF o EDF) che viene passata alla funzione di ordinamento come parametro. La funzione di ordinamento, a sua volta, si occupa di confrontare i prodotti in base alla priorità scelta e di riordinarli in base a tale criterio nella struttura dati stessa.

Per l'effettivo ordinamento dei prodotti, il software utilizza un algoritmo di sorting di tipo "insertion sort" che si è dimostrato il più efficiente per la dimensione dei dati in ingresso. Il secondo ordinamento, ovvero quello che compara il parametro "scadenza" in caso di parità di priorità o "priorità in caso di pari scadenza, è gestito da una funzione basata su algoritmo di bubble sort.

### 1.3.3 Output

Nel caso venisse specificato un file di output, il software stamperà su di esso i risultati della pianificazione secondo il seguente schema:

```
<Algoritmo di pianificazione scelto>:  
<ID del prodotto>:<Inizio produzione>  
[...]  
Conclusione:<timeslot termine produzione>  
Penalty:<penalità totale>
```

Esempio:

```
Pianificazione EDF:  
1:0  
4:2  
...  
9:55  
Conclusione: 63  
Penalty: 0
```

L'output della pianificazione viene comunque visualizzato a video.

### 1.3.4 Penalità

Se un prodotto non viene completato entro la scadenza, il programma calcola la penalità in base alla priorità del prodotto e alla quantità di slot temporali di ritardo. La penalità è calcolata come segue:

$$\text{Penalità} = \text{Priorità} \times \text{Ritardo}$$

## 1.4 Codice

### 1.4.1 Struttura della cartella di progetto

```
VR500751_VR504083
├── bin
│   └── *Pianificatore*
├── obj
├── Ordini
│   ├── EDF.txt
│   ├── NONE.txt
│   └── BOTH.txt
├── src
│   ├── AskInput.s
│   ├── AskOutput.s
│   ├── btoa.s
│   ├── BufferToArray.s
│   ├── EqualSort.s
│   ├── FinalData.s
│   ├── InsertionSort.s
│   ├── Intestazione.s
│   ├── menu.s
│   ├── main.s
│   ├── textin.s
│   └── Makefile
```

### 1.4.2 File sorgenti

**Il programma è composto dai seguenti file sorgente in linguaggio assembly:**

- AskInput.s: funzione per richiedere il nome del file di input all'utente;
- AskOutput.s: funzione per richiedere il nome del file di output all'utente;
- textin.s: funzione per la lettura di una scelta da tastiera.
- btoa.s: funzione loop per la conversione di un byte in ASCII;
- BufferToArray.s: funzione per la conversione del buffer di lettura in un array di struct;
- EqualSort.s: funzione per il secondo ordinamento dei prodotti;
- FinalData.s: funzione per la stampa dei risultati della pianificazione;
- InsertionSort.s: funzione per il primo ordinamento dei prodotti;
- Intestazione.s: scrittura dell'intestazione condizionale dell'output del programma;
- menu.s: file contenente la funzione per l'invocazione del menù utente;
- main.s: funzione principale del programma;

**Alcuni stralci di codice significativi:**

- Allocazione della memoria main.s:

```
130     malloc                # alloca spazio per l'array
131
132     movl len, %eax
133     sal $2, %eax           # multiplico per 4 (4byte per int) (perchè
                            # ho 4byte ogni riga/prodotto)
134     movl %eax, len
135
136     movl $45, %eax         # syscall per la malloc
137     xorl %ebx, %ebx        # alloco inizialmente 0byte
138     int $0x80
139
140     movl %eax, %esi        # salvo l'indirizzo di memoria in esi
                            # (zona di memoria puntata dalla malloc)
```



```

141
142     addl len, %eax           # aggiungo len a %eax (buffer allocazione)
143
144     movl %eax, %ebx          # salvo il risultato in ebx (lunghezza
                             # totale da allocare)
145     movl $45, %eax           # syscall per la malloc
146     int $0x80               # alloco la memoria - interrupt
147
148     cmpl %esi, %eax          # compara nuovo puntatore col vecchio (se
                             # sono uguali non ha allocato)
149     jle Error               # esi è l'indirizzo di memoria puntato
                             # dalla malloc (inizio array) mentre in eax c'è l'indirizzo di
                             # fine array
150
151     movl %esi, ArrayPointer # se tutto funziona salvo l'indirizzo di
                             # memoria in ArrayPointer così lo posso usare in BufferToArray

```

- Insertion sort insertionSort.s:

```

130     .section .data
131
132     key .int 0
133     base .int 0
134
135     .section .text
136     .global insertionSort
137     .type insertionSort, @function
138
139 insertionSort
140
141     pushl %ebp
142     movl %esp, %ebp
143
144     movl 8(%ebp), %edi # edi = Arr[fine]
145     movl 12(%ebp), %esi # esi = Arr[inizio]
146     movl 16(%ebp), %ecx # Metodo di sorting (+3 Priorità, +2 Scadenza)
147
148     popl %ebp
149
150     movl %ecx, base
151
152     xorl %ecx, %ecx # i = 0
153     xorl %edx, %edx # j = 0
154     addl base, %ecx # %ecx punta alla priorità/scadenza
155     addl $4, %ecx # %punta a 2 oggetto (1)
156
157 for
158
159     cmpl %ecx, %edi
160     jle endFor
161
162     movl (%esi, %ecx, 1), %eax
163     movb %al, key
164     subl base, %ecx
165     movl (%esi, %ecx, 1), %eax
166     push %eax
167     addl base, %ecx
168
169     movl %ecx, %edx # j = i
170     addl $-4, %edx # j = i - 1
171
172 while
173
174     cmpl $0, %edx
175     j1 PrepEndWhile

```

```

176
177     movb (%esi,%edx,1), %al
178
179     cmpb %al, key
180     jg PrepEndWhile
181
182     subl base, %edx
183     movl %edx, %eax
184     addl $4, %eax
185     movl (%esi,%edx,1), %ebx
186     movl %ebx, (%esi,%eax,1)
187     decl %edx
188
189     cmpl $3, base
190     je while
191
192     decl %edx
193
194     jmp while
195
196 PrepEndWhile
197
198     incl %edx
199
200     cmpl $3, base
201     je endWhile
202
203     incl %edx
204
205 endWhile
206
207     popl %eax
208     movl %eax, (%esi,%edx,1)
209
210     addl $4, %ecx
211
212     jmp for
213
214 endFor
215
216     ret

```

- Bubble sort equalSort.s:

```

130     .section .data
131
132     base .byte 0          # EDF or HPF
133     flag .byte 0         # 1 -> swap 0 -> no swap
134
135     .section .text
136     .global EqualSort
137     .type EqualSort, @function
138
139 EqualSort
140
141     pushl %ebp
142     movl %esp, %ebp
143
144     movl 8(%ebp), %edi     #len
145     movl 12(%ebp), %esi   #pointer
146     movl 16(%ebp), %ecx   #Base
147
148     popl %ebp             #restore stack
149
150     movb %cl, base        #salvo base (EDF or HPF)

```

```

151     movl %ecx, %eax          # i
152     movl %eax, %ebx          # j
153     addl $4, %ebx            # j + 1
154
155 loop
156
157     cmpl %eax, %edi           #len <= i (uscita)
158     jle endLoop
159
160     xorl %ecx, %ecx
161     xorl %edx, %edx
162
163     movb (%esi, %eax, 1), %cl
164     movb (%esi, %ebx, 1), %dl
165
166     cmpb %cl, %dl             #confronto i e j
167     je Check                  #se sono uguali vado a check
168
169     jmp next
170
171 Check
172
173     cmpb $3, base             #se base = HPF (priorità) -> decrement
174     je decrement
175
176 increment                     # guardo priorità
177
178     incl %eax                  #incremento i
179     incl %ebx                  #incremento j
180
181     movb (%esi, %eax, 1), %cl    # preparo il confronto dei
    prossimi valori
182     movb (%esi, %ebx, 1), %dl
183
184
185     subl $3, %eax              # torno a puntare l'inizio dell'elemento
186     subl $3, %ebx              # =
187
188     cmpb %cl, %dl              # confronto i e j
189     jl reset
190     je SetDurata
191
192     movl (%esi, %eax, 1), %ecx
193     movl (%esi, %ebx, 1), %edx
194
195     movl %ecx, (%esi, %ebx, 1)
196     movl %edx, (%esi, %eax, 1)
197
198     movb $1, flag
199
200     jmp reset
201
202 decrement                     # guardo scadenza
203
204     decl %eax
205     decl %ebx
206
207     movb (%esi, %eax, 1), %cl
208     movb (%esi, %ebx, 1), %dl
209
210
211     subl $2, %eax
212     subl $2, %ebx
213

```

```

214     cmpb %dl, %cl
215     jg reset
216     je SetDurata
217
218     movl (%esi, %eax, 1), %ecx
219     movl (%esi, %ebx, 1), %edx
220
221     movl %ecx, (%esi, %ebx, 1)
222     movl %edx, (%esi, %eax, 1)
223
224     movb $1, flag
225
226     jmp reset
227
228 SetDurata          # se scadenza uguale priorità uguali, controllo la
durata
229
230     addl $1, %eax    # incremento così vado da ID a durata
231     addl $1, %ebx
232
233 Durata
234
235     movb (%esi, %eax, 1), %cl
236     movb (%esi, %ebx, 1), %dl
237
238
239     subl $1, %eax
240     subl $1, %ebx
241
242     cmpb %cl, %dl
243     jge reset
244
245     movl (%esi, %eax, 1), %ecx
246     movl (%esi, %ebx, 1), %edx
247
248     movl %ecx, (%esi, %ebx, 1)
249     movl %edx, (%esi, %eax, 1)
250
251     movb $1, flag
252
253 reset
254
255     addb base, %al    # resettato base
256     addb base, %bl    # resettato base
257
258 next
259
260     cmpb $1, flag
261     je Redo
262
263     addl $4, %eax
264     addl $4, %ebx
265     jmp loop
266
267 Redo          # se ho fatto uno swap, ripeto il confronto
268
269     movl $0, flag
270     movl base, %ecx
271     movl %ecx, %eax
272     movl %eax, %ebx
273     addl $4, %ebx
274
275     jmp loop
276

```

```

277 endLoop
278
279     ret

```

- Loop scrittura FinalData.s:

```

130     .section .data
131
132 PenaltyMsgLen .byte 9
133 TimeMsgLen .byte 13
134
135 PenaltyMsg .string "Penalty: "
136 TimeMsg .string "Conclusione: "
137
138 timeAscii .byte 4
139 penaltyAscii .byte 6
140
141
142 .section .text
143     .globl finalData
144     .type finalData, @function
145
146 finalData
147
148     pushl %ebx #salvo la penalita'
149     pushl %eax #salvo il tempo
150     addl %edx, %edi # aggiungo la lunghezza del buffer all'inizio del
151         buffer per avere il primo spazio libero
152     xorl %ecx, %ecx #per sicurezza azzero ecx
153
154     movl $TimeMsg, %esi
155     movb TimeMsgLen, %cl
156     cld
157     LoopTime          #ctrl c e ctrl v
158     lodsb
159     stosb
160     loop LoopTime
161
162 getTime #Tempo in numeri in %ebx e dove scrivere in %edi
163
164     popl %eax #Riprendo dalla stack il tempo
165     movl $10, %ebx
166     leal timeAscii+3, %esi #un semplicissimo btoa che trasforma il
167         tempo in ascii
168     movl $10, (%esi)
169
170 loopTime
171
172     xorl %edx, %edx
173     divl %ebx          # divido per prendere il resto
174
175     addb $48, %dl      #aggiungo 48 per trasformare il numero in
176         ascii
177     decl %esi          #decremento il puntatore che fino a questo
178         punto punta a 10 (vedi sopra)
179     movb %dl, (%esi)
180
181     test %eax, %eax
182     jz BufferTime
183     jmp loopTime
184
185 BufferTime #copio il buffer in %edi(puntatore al buffer)

```

```

185     movb (%esi), %al
186     cmpb $10, %al
187     je next
188     movb %al, (%edi)
189
190     incl %edi
191     incl %esi
192
193     jmp BufferTime
194
195 next    #finito il tempo uso il tappo \n
196
197     movb $10, (%edi)
198     incl %edi
199
200     xorl %ecx, %ecx
201
202     movl $PenaltyMsg, %esi
203     movb PenaltyMsgLen, %cl
204     cld                                     # ctrl c e ctrl v
205     LoopPenalty
206         lodsb
207         stosb
208         loop LoopPenalty
209
210 getPenalty
211
212     popl %eax #Riprendo dalla stack la penalita'
213     movl $10, %ebx
214     leal penaltyAscii+5, %esi
215     movl $10, (%esi)
216
217 loopPenalty
218
219     xorl %edx, %edx
220     divl %ebx
221
222     addb $48, %dl
223     decl %esi
224     movb %dl, (%esi) #stessa cosa di sopra ma per la penalita'
225
226     test %eax, %eax
227     jz BufferPenalty
228
229     jmp loopPenalty
230
231 BufferPenalty
232
233     movb (%esi), %al
234     cmpb $10, %al
235     je end
236     movb %al, (%edi)
237
238     incl %edi
239     incl %esi
240
241     jmp BufferPenalty
242
243 end
244
245     movb %al, (%edi)    #aggiungo \n
246
247     ret

```

**In allegato si troveranno i seguenti dataset per testare il programma:**

- `Makefile`: file per la compilazione del programma;
- `EDF.txt`: file di input con penalità uguale a zero con EDF e maggiore di zero con HPF;
- `BOTH.txt`: file di input con penalità uguale a zero con entrambi gli algoritmi;
- `NONE.txt`: file di input con penalità maggiore di zero con entrambi gli algoritmi.

## **1.5 Scelte progettuali**

### **1.5.1 Output su file / funzioni extra-specifiche**

Durante la programmazione del software sono state implementate alcune funzionalità aggiuntive:

- Salvataggio dei risultati della pianificazione su file;
- Visualizzazione del menù utente multi-scelta;
- Possibilità di lanciare il software senza specificare un file di output.
- Modifica a runtime di file di input e output.
- Visualizzazione di un messaggio di errore nel caso di file di output non specificato.

### **1.5.2 Algoritmi di sorting**

Per l'ordinamento dei prodotti, sono stati implementati due algoritmi di sorting differenti:

- **Sorting principale**: ordinamento basato su algoritmo 'insertion sort' per il confronto diretto tra il valore indicato dalla tipologia di pianificazione (EDF: deadline, HPF: priorità);
- **Sorting secondario**: ordinamento secondario basato su algoritmo di tipo 'bubble sort' per il confronto in caso di valore principale identico.

### **1.5.3 Limiti del software**

Durante la fase di testing sono sorte alcune limitazioni del software:

- Il software non è in grado di gestire generalmete prodotti con valori maggiori di 256;
- Il software non è in grado di gestire file di input aventi dimensione maggiore di 1024 Byte.
- Il software non è in grado di gestire file di output aventi dimensione maggiore di 1024 Byte.