Considerăm proiectele ordonate crescător după ziua finală şi pentru fiecare proiect i notăm cu *ult(i)* cel mai mare indice j $(1 \leq j < i)$ al unui proiect care nu se suprapune cu proiectul i sau 0 dacă nu există nici un astfel de proiect.

Pentru datele de intrare din enunţ, după sortarea lor, vom obţine următoarele valori pentru *ult*:

1.	Java4All	1	3	250	$ult(1) = 0$
2.	FileSeeker	2	6	650	$ult(2) = 0$
3.	BlackFace	4	12	800	$ult(3) = 1$
4.	NiceTry	7	13	850	$ult(4) = 2$
5.	OzWizard	4	16	900	$ult(5) = 1$
6.	JustDoIt	13	18	1000	$ult(6) = 4$
7.	BestJob	15	22	900	$ult(7) = 4$
8.	Test2Test	25	27	300	$ult(8) = 7$

Notând cu $bm[i]$ bonusul maxim care se poate obţine prin planificarea primelor i proiecte $(0 \leq i \leq n)$ şi considerând bonusul asociat unui proiect i ca fiind notat cu $c[i]$, vom obţine următoarea relaţie de recurenţă:

$$bm[i] = \begin{cases} 0, & \text{dacă } i = 0 \\ \max\{bm[i-1], c[i] + bm[ult(i)]\}, & \text{dacă } i \geq 1 \end{cases}$$

Relaţia de recurenţă ne "spune" faptul că un proiect i va fi planificat (singur sau alipit după un şir de alte proiecte planificate anterior) doar în cazul în care astfel se obţine un bonus mai mare decât neplanificându-l. Atenţie, în unele cazuri este mai rentabil să planificăm un anumit proiect i (singur, "pierzând" toate proiectele planificate anterior, sau alipit la un şir de alte proiecte planificate anterior), iar în alte cazuri este mai rentabil să nu-l planificăm deloc (singur sau chiar alipit de alte proiecte planificate anterior) deoarece bonusul obţinut ar fi mai mic!

Practic, pentru fiecare proiect i vom proceda astfel:

1. Calculăm *ult(i)*, adică verificăm dacă există sau nu un alt proiect planificat anterior după care am putea să programăm proiectul curent. Astfel, obținem următoarele două cazuri:

   1.1. nu există nici un proiect după care să putem planifica proiectul i, deci $ult(i) = 0 \Rightarrow$ bonusul asociat proiectului i va fi $c[i] + bm[0] = c[i]$
   1.2. există un proiect după care să putem planifica proiectul i, deci ult(i) $\neq 0 \Rightarrow$ bonusul asociat proiectului i va fi $c[i] + bm[ult(i)]$

2. Comparăm bonusul asociat proiectului i, egal cu $c[i] + bm[ult(i)]$ și calculat la pasul anterior, cu bonusul maxim obținut prin planificarea primelor $i - 1$ proiecte și reținem maximul dintre ele (practic, decidem dacă planificăm proiectul curent sau nu).

Observație: Valorile *ult(i)* vor fi folosite pentru a obține modalitatea optimă de planificare a proiectelor (reconstituirea soluției)!

i	1	2	3	4	5	6	7	8
	P3	P7	P1	P5	P8	P6	P4	P2
p	1 3	2 6	4 12	7 13	4 16	13 18	15 22	25 27
	250	650	800	850	900	1000	900	300

ult	0	0	1	2	1	4	4	7

bm	0	250	650	1050	1500	1500	2500	2500	2800
		0 250	250 650	650 250+800	1050 650+850	1500 250+900	1500 1500+1000	2500 850+900	2500 2500+300

```
typedef struct
{
   char nume_proiect[50];
   int ziua_initiala , ziua_finala , bonus;
}Proiect;

int cmpProiecte(const void *p1, const void *p2)
{
   Proiect vp1 = *(Proiect *)p1;
   Proiect vp2 = *(Proiect *)p2;

   return vp1.ziua_finala - vp2.ziua_finala;
}
```

```
……….……

   qsort(p + 1, n, sizeof(Proiect), cmpProiecte);

   bm[0] = 0;
   for (i = 1; i <= n; i++)
   {
      ult[i] = 0;

      for (j = i-1; j >= 1; j--)
         if (p[j].ziua_finala <= p[i].ziua_initiala)
         {
            ult[i] = j;
            break;
         }

      if(p[i].bonus + bm[ult[i]] > bm[i-1])
         bm[i] = p[i].bonus + bm[ult[i]];
      else
         bm[i] = bm[i-1];
   }
```

```
fout = fopen("proiecte.out", "w");

  i = n;
  while(i >= 1)
     if(bm[i] != bm[i-1])
     {
        fprintf(fout, "%s %d %d %d\n", p[i].nume_proiect, p[i].ziua_initiala,
                                                  p[i].ziua_finala, p[i].bonus);
        i = ult[i];
     }
     else
        i--;

  fprintf(fout, "\nBonusul echipei: %d\n", bm[n]);

  fclose(fout);
```