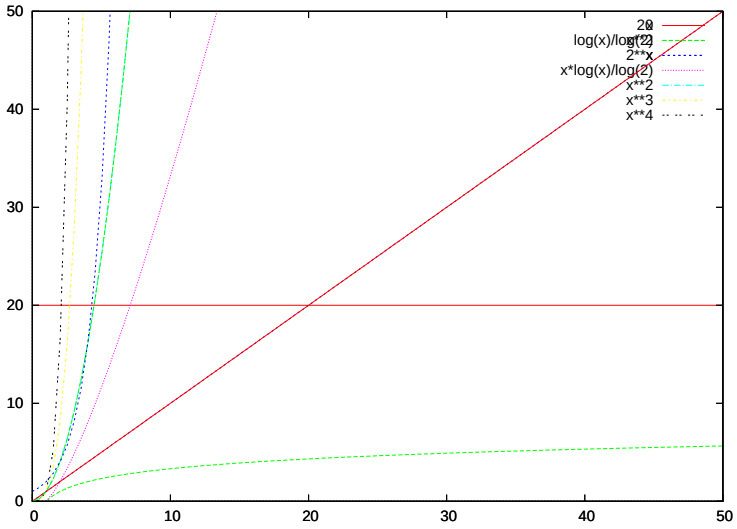

FIG. 1.2 – Représentation graphique de quelques fonctions usuelles

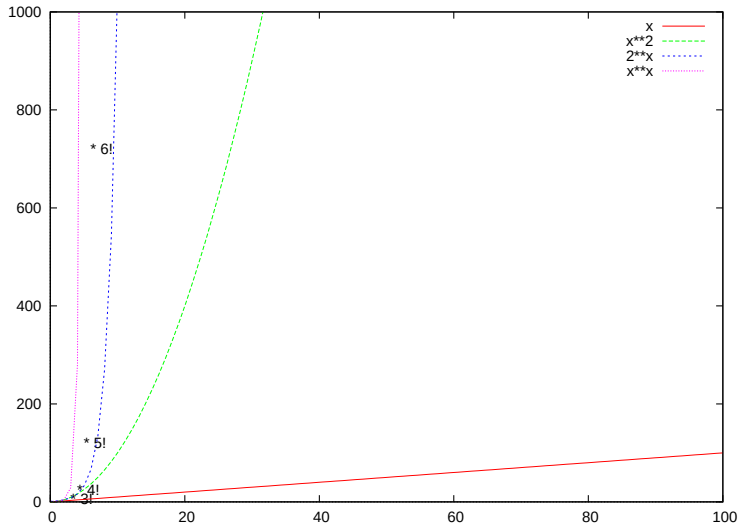

FIG. 1.3 – Représentation graphique des fonctions de grande complexité

$f(n)$	10	1 000	100 000	1 000 000
$\log_2 n$	0.000003 s	0.000010 s	0.000017 s	0.000020 s
n	0.000001 s	0.001 s	0.1 s	1 s
$n \log_2 n$	0.000002 s	0.01 s	1.7 s	20 s
n^2	0.0001 s	1 s	2 h 45	11 jours 1/2
n^3	0.001 s	17 min	32 siècles	30 000 siècles
2^n	0.001 s	10^{285} siècles	10^{10^4} siècles	10^{10^5} siècles

TAB. 1.1 – Quelques "*run times*" associés à des complexités courantes

1.4 Quelques réflexions sur la complexité à partir d'exemples

Parcours

```
1. for (i=1 ; i<length(t) ; i++)
2.    print t[i]
```

```
1. for (i=1 ; i<n ; i++)
2.    read(a) ;
3.    b = a*9/5 + 32 ;
4.    print b ;
```

Notions : – taille de l'input
– *random-access machine*
– temps d'exécution – fonction linéaire *vs.* quadratique

Recherche (linéaire)

```
1. for (i=1 ; i<length(t) ; i++)
2.     if (t[i]==key) print(i) ;
```

```
1. i= 1
2. while(t[i] <> key)
3.    i = i + 1 ;
4. if (t[i]==key) print i ;
```

Notions : – cas le pire et cas moyen
– taux (ordre) de croissance

Tri

```
champion:
1. best = t[1] ;
2. for (i=2 ; i<length(t) ; i++)
3.   if (best < t[i]) best = t[i]
```

```
champion(d,f) :
1. best = t[d] ;
2. for (i=d+1 ; i<f ; i++)
3.   if (best < t[i]) best = t[i]
4. return best
```

```
tri :
1. for (i=1 ; i<length(t) ; i++)
2.   t[i] = champion(i,length(t)) ;
```