Institut supérieur d'informatique et de technologie de communication.

Algorithme Avance

Projet 2

Les Algo de tri

- ***Réalisé par: Bacem Abroug***
- ***Classe: 2DNI_G2.***

Introduction:

Lorsque vous jouez aux cartes (tarot, belote, rami, etc.), vous avez plusieurs cartes dans votre main. Afin de mieux voir les cartes et de pouvoir optimiser leur stratégie, la plupart des joueurs ordonnent par couleur et par valeur les cartes, en fonction du jeu auquel ils jouent. Les règles du jeu vous indiquent un ordre dans lequel il est possible de ranger vos cartes. Comment procédez-vous ? Il n'y a pas une seule technique. Certains joueurs sont plus rapides que d'autres pour ranger leur jeu. Quel est leur secret ?

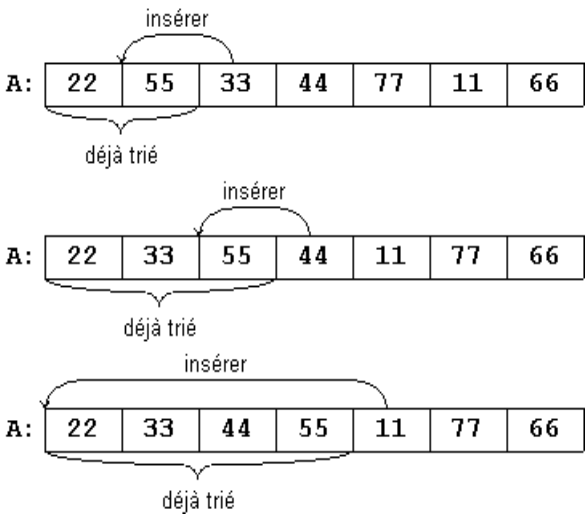
Ordonner, ranger, trier un nombre considérable de données sont des opérations fastidieuses pour un être humain, alors qu'un ordinateur est bien plus efficace dans ces tâches répétitives. Il existe de nombreuses manières de programmer un ordinateur pour trier des données. Ainsi, les plus grands informaticiens se sont penchés sur la question afin d'élaborer des algorithmes de tri les plus efficaces possibles. De nos jours, cette activité est croissante à cause de l'explosion des données disponibles via Internet. Un des principaux défis des années à venir est la manipulation des données issues du "BIG DATA" généré par l'interconnectivité des objets de notre quotidien.

Nous commençons par présenter cinque méthodes de tri classiques,puis en va exprime l'algorithm de chaqu'une methode.Enfin on va calculer la complexite de chaque algorithme et donne une comparison on se basant sur la complexite

I. Les Algorithmes de tri

I.1 Tri par insertion:

Intuition : Une méthode de tri très simple correspond à ce que font certains joueurs de cartes en prenant les cartes une à une et en les rangeant au fur et à mesure dans leur main. Cet algorithme est connu sous le nom de tri par insertion : partant d'un ensemble ordonné de cartes, chaque nouvelle carte est correctement insérée à sa place relativement aux cartes déjà présentes.



Algorithm:

```
algorithme tri_insert(vect, ordre←'croi')
   si ordre = 'croi'
      pour counter dans range(len(vect)-1)
         x ← vect[counter+1]
         j ← counter
         Tant que ((vect[j] > x) et (j ≥ 0))
            vect[j+1] ← vect[j]
            diminuer j de  1
         vect[j+1] ← x
   sinon, si  ordre = 'dec'
      pour counter dans range(len(vect)-1)
         x ← v[counter+1]
         j ← counter
         Tant que ((vect[j] < x) et (j ≥ 0))
            vect[j+1] ← vect[j]
            diminuer j de  1
         vect[j+1] ← x
   sinon
      afficher ('entre lordre de trie soit croi pour croissant ou
dec pour decorissant')

   renvoyer vect
```

Complexité:

En temps :

dans le cas le pire, on fait, dans la boucle intérieure, i –1 comparaisons et autant de décalages : temps A(i –1) + B ;

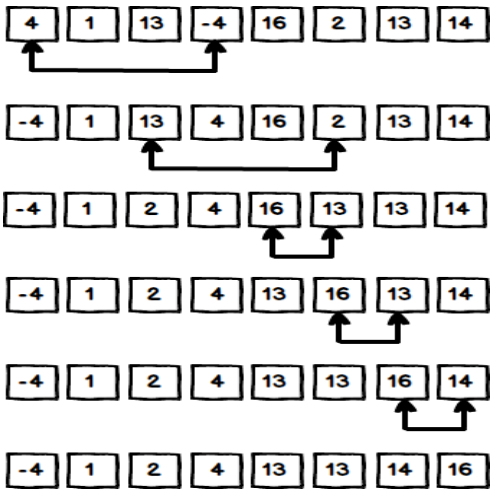
cette boucle est parcourue pour i = 2, ... , n.

Le temps total dans le cas le pire est donc O(n^2).

I.2 Tri par selection:

Intuition : Le tri par minimum isole le minimum parmi les nombres et le place au début de la liste en l'échangeant avec le nombre en dernère position, puis recommence avec les nombres restants. En répétant ce processus autant de fois qu'il y a de nombres nous obtenons les nombres triés du plus petit au plus

grand. Il est possible de faire la même chose en cherchant le maximum à chaque fois afin d'obtenir les nombres dans l'ordre décroissant.



Algorithm:

```
algorithme tri_select(vect,)

      i ← 0
      Tant que i < len(vect)

         x ← vect[i]
         pour count dans range(i+1, len(vect))

            si x > vect[count]
               x ← vect[count]
               vect[count] ← vect[i]
               vect[i] ← x
         fin pour

         i← i+1
         fin Tq
Fin algorithme tri_select
```

Complexité:

Complexité de l'algorithme.

En temps : la boucle intérieure demande un temps A(n – i) + B

cette boucle est parcourue pour i = 1, 2, ... , n –1.

Le temps total est donc An(n –1)/2 + Bn = O(n^2).

=>Toutes les opérations sont à effectuer quel que soit le tableau.

I.3 Tri a Bull:

Intuition : Le principe du tri à bulles est de parcourir la liste en échangeant lors du parcours deux éléments consécutifs s'ils sont rangés dans le mauvais ordre et de répeter ce processus jusqu'à ce qu'il n'y ait plus d'échanges lors d'un parcours. Dans ce cas la liste est triée.

Algorithm:

```
algorithme tri_bul(vect)
 permut ← vrai
   p ← len(vect)-1

      Tant que permut
         permut ← faux
         pour counter allant de 0 à p-1
            si vect[counter] > vect[counter+1]
               x ← vect[counter]
               vect[counter] ← vect[counter+1]
               vect[counter+1] ← x
               permut ← vrai
         fin pour
         p=p-1
         i=i+1
         fin TQ
```

Complexité:

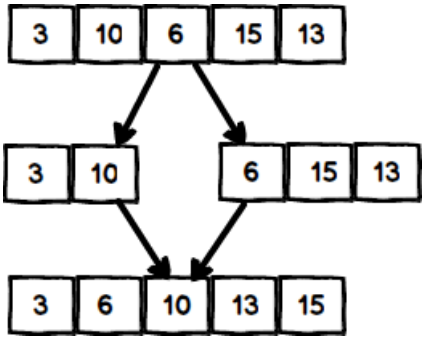
En temps : Le raisonnement est le même que pour le tri par insertion.

Le temps total dans le cas le pire est donc O(n^2).

Si on trie une permutation, le nombre d'échanges faits est le nombre d'inversions. Le nombre de comparaisons est au moins ce nombre. Or le nombre moyen d'inversions est en O(n^2)

I.4 Tri fusion:

Intuition : Un algorithme plus rapide que les deux précédents est le tri par fusion. Ce tri repose sur les deux remarques suivantes :



– Un ensemble d'une carte est nécessairement ordonné.

– Il est facile de fusionner en une série ordonnée deux séries déjà triées. Pour cela il suffit de comparer la première carte de chaque série, de ranger la plus petite des deux dans une nouvelle série et de recommencer sur ce qui reste des deux séries initiales.

Algorithm:

```
algorithme tri_fusion(w)
   algorithme triFussion(v)

      si len(v) > 1
         mid ← len(v)//2
         tourner vers la gauche de ← v[mid]
         tourner vers la droite de ← v[mid]

         triFussion(tourner vers la gauche de)
         triFussion(tourner vers la droite de)

         i ← 0
         j ← 0
         k ← 0
         Tant que i < len(tourner vers la gauche de) et j <
len(tourner vers la droite de)
            si tourner vers la gauche de[i] ≤ tourner vers la
droite de[j]
               v[k] ← tourner vers la gauche de[i]
               i ← i+1
            sinon
               v[k] ← tourner vers la droite de[j]
               j ← j+1
            k ← k+1

         Tant que i < len(tourner vers la gauche de)
            v[k] ← tourner vers la gauche de[i]
            i ← i+1
            k ← k+1

         Tant que j < len(tourner vers la droite de)
            v[k] ← tourner vers la droite de[j]
            j ← j+1
            k ← k+1

   triFussion(w)
   renvoyer w
```

Complexité:

Pour simplifier, supposons que la taille du tableau initial est une puissance de 2 : n = 2^i .

- fusion demande`a chaque appel au plus n/2 comparaisons.
- C(n) <=2 × C (n/2) + n/2

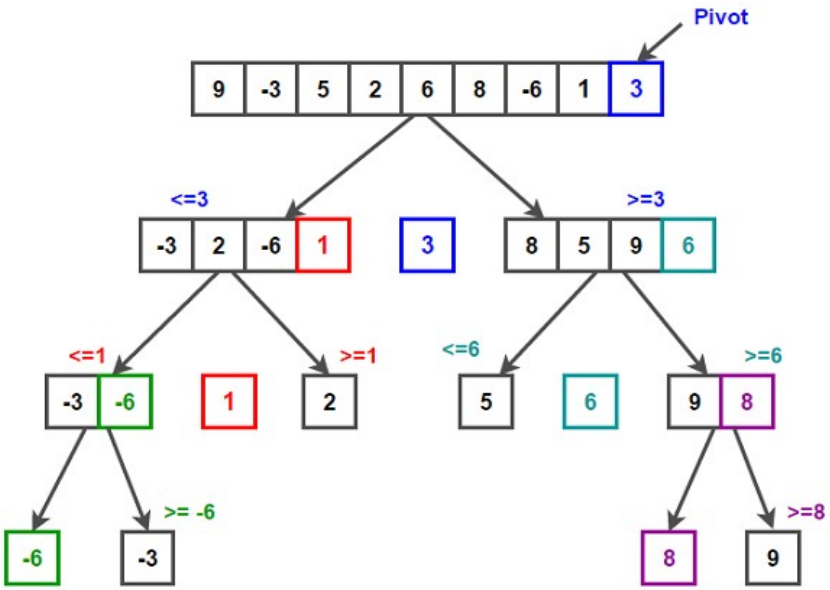
C(n) <=2 × [2C(n/2^2) + n/2^2] + n/2

C(n) <=k × n + 1/2 nlog2(n) <= Kn log(n)

Le temps total est donc O(nlog(n))

I.5 Tri Rapide:

Intuition : L'idée de cet algorithme est de choisir un élément appelé pivot et de le ranger à sa place définitive, triant tous les éléments inférieurs au pivot à sa gauche et tous ceux qui sont supérieurs au pivot à sa droite. Plus précisément cet algorithme fonctionne comme suit : – choisir un élément p au hasard parmi les éléments à trier. Cet élément est appelé le pivot. – séparer les éléments restants en deux listes, d'une part ceux qui sont plus petits que p, d'autre part ceux qui sont plus grands. – trier chacune des listes obtenues en utilisant ce même algorithme. – aggréger les deux listes triées, en les mettant bout à bout.



Algorithm:

```
algorithme trirapide(L)
    """trirapide(L) tri rapide (quicksort) de la liste L"""
    algorithme trirap(L, g, d)
        pivot ← L[(g+d)//2]
        i ← g
        j ← d
        Tant que vrai
            Tant que L[i] < pivot
                augmenter i de  1
            Tant que L[j] > pivot
                diminuer j de  1
            si i > j
                break
            si i < j
                L[i], L[j] ← L[j], L[i]
            augmenter i de  1
            diminuer j de  1
        si g < j
            trirap(L, g, j)
        si i < d
            trirap(L, i, d)

    g ← 0
    d ← len(L)-1
    trirap(L, g, d)
```

Complexité:

Pour ce qui est du temps nécessaire, celui de la fonction partage est proportionnel à j – i quel que soit p. Donc, si la liste est déjà classée et si les pivots sont successivement 1, 2, ..., n, le temps nécessaire est quadratique.

Par contre, en moyenne, le temps est en n log n. Intuitivement, le pivot correspondra à une valeur centrale dans l'intervalle choisi du tableau. Par conséquent, les intervalles sur lesquels on itère vaudront en moyenne n/2. On se retrouve alors dans une situation de type "diviser pour régner". D'où le temps moyen en O(n log n).

II. Comparaison

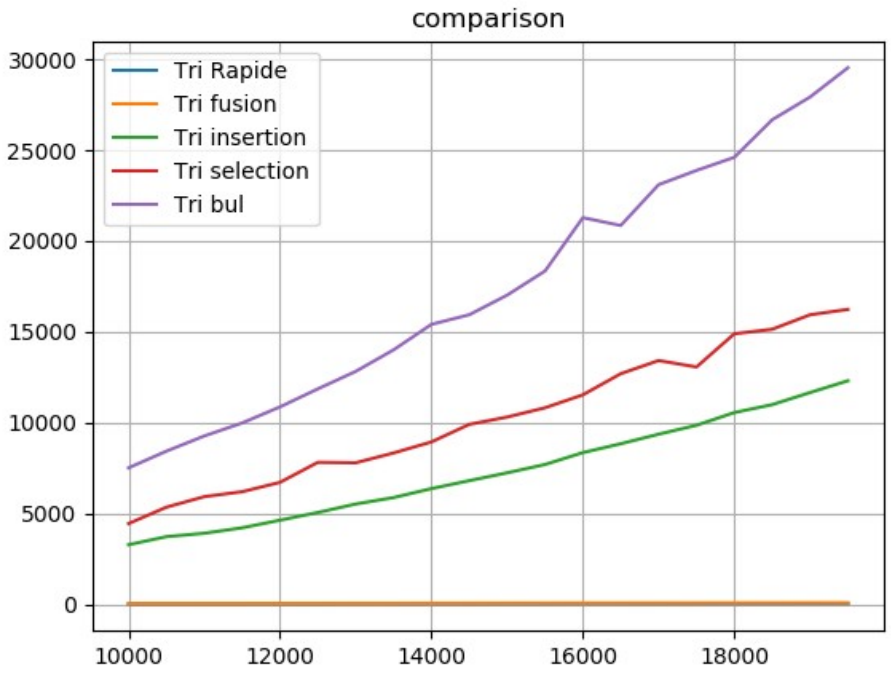
Complexite:

Methode	Complexite
Tri par selection	O(n^2)
Tri a bulle	O(n^2)
Tri par insertion	O(n^2)
Tri fusion	O(nlog(n))
Tri rapide	O(nlog(n))

Temps d'execution

	N=10000	N=15000	N=19500
Tri par selection	10002ms	26538ms	30853ms
Tri a bulle	8802ms	20556ms	32840ms
Tri par insertion	3400ms	9432ms	15850ms
Tri fusion	57ms	65ms	87ms
Tri rapide	21ms	29ms	39ms

Courbe:



III. Conclusion

L’importance des algorithmes de tri dans notre quotidien n’est plus à démontrer, la moindre requête à votre site préféré nécessite de trier les résultats pour les afficher. Une motivation plus mathématique est le calcul de la médiane d’une suite de valeurs qui est immédiate sur une liste triée. Remarquons qu’il existe pour le calcul de la médiane un algorithme naïf moins efficace qui consiste à retirer le maximum et le minimum de la liste jusqu’à obtenir un ou deux éléments.