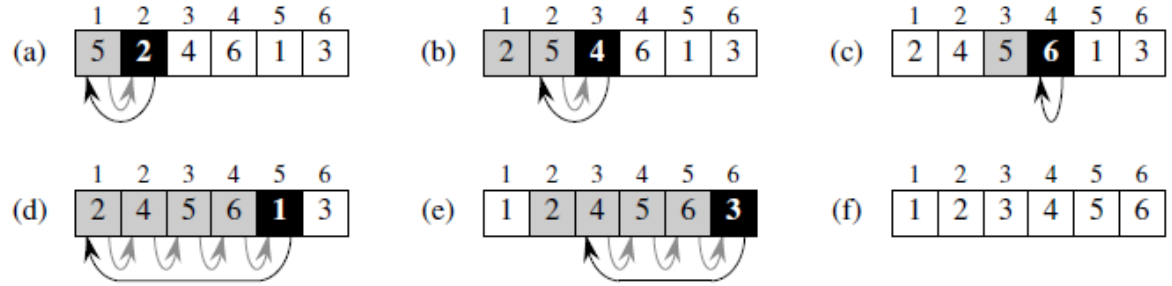
Université Assane Seck de Ziguinchor
UFR sciences et Technologies
Département Informatique

Exercices de TD (Feuille 2)

Consignes : Les exercices mis en évidence sont obligatoires pour valider le chapitre 2. Les autres exercices sont optionnels. Il vous est conseillé de les faire après les exercices obligatoires.

2.1.1 À l'aide de la figure suivante, illustrer l'action de TRI-INSERTION sur le tableau A = « 31, 41, 59, 26, 41, 58 ».



2.1.2 Réécrire la procédure TRI-INSERTION pour trier dans l'ordre non croissant et non dans l'ordre non décroissant.

2.1.3 Considérez le ***problème de la recherche*** :

Entrée : Une suite de n nombres A = « $a1, a2, \ldots, an$ » et une valeur v.

Sortie : Un indice i tel que $v = A[i]$, ou bien la valeur spéciale NIL si v ne figure pas dans A.

Écrire du pseudo code pour ***recherche linéaire***, qui parcoure la suite en cherchant v. En utilisant un invariant de boucle, montrer la validité de l'algorithme. Vérifier que l'invariant possède bien les trois propriétés requises.

2.1.4 On considère le problème consistant à additionner deux entiers en représentation binaire stockés sur n bits, rangés dans deux tableaux A et B à n éléments. La somme des deux entiers doit être stockée sous forme binaire dans un tableau C à $n + 1$ éléments. Énoncer le problème formellement et écrire du pseudo-code pour additionner les deux entiers.

2.2.1 Exprimer la fonction $n^3/1000 - 100n^2 - 100n + 3$ à l'aide de la notation Θ.

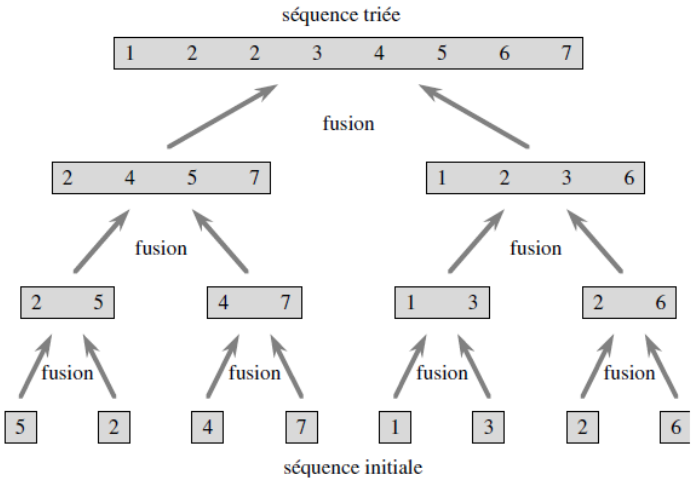
2.2.2 On considère le tri suivant de n nombres rangés dans un tableau A : on commence par trouver le plus petit élément de A et on le permute avec $A[1]$. On trouve ensuite le deuxième plus petit élément de A et on le permute avec $A[2]$. On continue de cette manière pour les $n-1$ premiers éléments de A. Écrire du pseudo code pour cet algorithme, connu sous le nom de ***tri par sélection***. Quel est l'invariant de boucle de cet algorithme ? Pourquoi suffit-il d'exécuter

l'algorithme pour les $n-1$ premiers éléments ? Donner les temps d'exécution associés au cas optimal et au cas le plus défavorable en utilisant la notation **Θ**.

2.2.3 On considère une fois de plus la recherche linéaire (voir exercice 2.1.3). Combien d'éléments de la séquence d'entrée doit-on tester en moyenne, si l'on suppose que l'élément recherché a une probabilité égale d'être l'un quelconque des éléments du tableau ? Et dans le cas le plus défavorable ? Quels sont les temps d'exécution du cas moyen et du cas le plus défavorable, exprimés avec la notation **Θ**? Justifier les réponses.

2.2.4 Comment modifier la plupart des algorithmes pour qu'ils aient un bon temps d'exécution dans le cas le plus favorable ?

2.3.1 En s'inspirant de la figure suivante, illustrer le fonctionnement du tri par fusion sur le tableau A = « 3, 41, 52, 26, 38, 57, 9, 49 ».



2.3.2 Réécrire la procédure FUSION de telle sorte qu'elle n'emploie pas de sentinelles mais qu'à la place elle s'arrête quand l'un des deux tableaux L et R a eu tous ses éléments copiés dans A, en copiant alors le reste de l'autre tableau dans A.

2.3.3 Utiliser l'induction mathématique pour montrer que, lorsque n est une puissance exacte de 2, la solution de la récurrence

$$T(n) = \begin{cases} 2 & \text{si } n = 2, \\ 2T(n/2) + n & \text{si } n = 2^k, \text{ pour } k > 1 \end{cases}$$

est T(n) = n lg n.

2.3.4 Le tri par insertion peut être exprimé sous la forme d'une procédure récursive de la manière suivante. Pour trier $A[1 \ldots n]$, on trie récursivement $A[1 \ldots n-1]$ puis on insère $A[n]$

dans le tableau trié $A[1 . . n - 1]$. Écrire une récurrence pour le temps d'exécution de cette version récursive du tri par insertion.

2.3.5 En reprenant le problème de la recherche (voir exercice 2.1.3), observez que, si la séquence A est triée, on peut comparer le milieu de la séquence avec v et supprimer la moitié de la séquence pour la suite des opérations. La ***recherche dichotomique*** est un algorithme qui répète cette procédure, en divisant par deux à chaque fois la taille de la partie restante de la séquence. Écrire le pseudo code, itératif ou récursif, de la recherche dichotomique. Expliquer pourquoi le temps d'exécution de la recherche dichotomique, dans le cas le plus défavorable, est $\Theta(\lg n)$.

2.3.6 On observe que la boucle **tant que** des lignes 5 – 7 de la procédure TRI-INSERTION de la section 2.1 utilise une recherche linéaire pour parcourir (à rebours) le sous-tableau trié $A[1 . . j - 1]$. Est-il possible d'utiliser à la place une recherche dichotomique (voir l'exercice 2.3.7) pour améliorer le temps d'exécution global du tri par insertion, dans le cas le plus défavorable, de façon qu'il devienne $\Theta(n \lg n)$?

2.3.7 Décrire un algorithme en $\Theta(n \lg n)$ qui, étant donnés un ensemble S de n entiers et un autre entier x, détermine s'il existe ou non deux éléments de S dont la somme vaut exactement x.