Survol de la matière

**Qu'est-ce qu'une opération de base?**

*C ‘est une opération qui prend un temps constant, comme diviser et multiplier par 10, par exemple.*

**Qu'est-ce qu'un algorithme correct? Efficace?**

*Un algorithme est correct s’il fait ce qu‘il est censé de faire, il est exact s’il donne la bonne sortie pour toute les entrées possibles (il ne se trompe jamais!). Il est efficace s’exécute en peu d’opérations basiques.*

**Quelles sont les trois sections de mémoire pour les exécutables? Quels sont leur rôle, leur contenu? Quand sont-elles libérées? Déclarez des variables dans chaque section.**

*La pile d’exécution : c’est une structure de données qui suit le principe dernier arrivé, premier servi. L’exécution se fait à la fois avec un pointeur d’exécution. La pile contient l’état d’exécution du programme. Les variables dans la pile sont libérées soit au return ou à la fin du programme.*

*Ex de déclaration : int i = 0; (variables locales/formelles)*

*Le tas : n’a pas une structure particulière, sa taille s’ajuste au cours de l’exécution et la mémoire est dynamique. Il permet au programmeur de contrôler la mémoire. Les variables dans le tas sont libérées quand on appelle le free() pour la variable ou à la fin du programme.*

*Ex de déclaration : float\* tab = (float\*) malloc(sizeof(float) \* 8); pour créer un tableau de 8 cases*

*float\* tab = (float\*) calloc(8, sizeof(float)); pour créer un tableau de 8 cases initialisé a 0; (malloc/calloc)*

*Le data : est rapide, compacte, statique et sa taille est fixée à la compilation. Contient, entre autres, le code, les instructions et les variables globales. La mémoire est libérée à la fin du programme.*

**Qu'est-ce qu'une pile? Une file? Une liste chaînée? Un tableau dynamique?**

*Une pile (stack) c’est une structure de données qui suit le principe dernier arrivé, premier servi. L’exécution se fait à la fois avec un pointeur d’exécution. La pile contient l’état d’exécution du programme.*

*En C, une file est une liste chaînée où chaque élément pointe vers le suivant. Le dernier élément de la file point vers NULL (il n’a pas de suivant).*

*Une liste chaînée est une structure ordonnée qui représente une collection ordonnée de taille variable d’éléments de même type, dont chaque élément pointe vers son suivant. Si la liste est doublement chaînée, chaque élément pointe vers son suivant et son précédent.*

*Un tableau dynamique est un tableau qui on déclare avec un malloc/calloc dans le tas et on peut changer sa taille pendant l’exécution du programme avec un realloc pour qu’il soit plus grand ou plus petit au besoin.*

**Comment ajoute/retire-t-on un élément dans un tableau dynamique? Une liste chaînée?**

*Pour ajouter ou retirer des éléments dans un tableau dynamique, on utilise realloc.*

*Dans une liste chaînée, on peut ajouter, lire, modifier et supprimer des nœuds à condition qu’on connaisse l’adresse du premier nœud. L’ajoute et la suppression autour d’une adresse connue sont optimales, il faut juste changer la référence pour le suivant->suivant ou NULL.*

**Appliquez les quatre algorithmes de tri (sélection/insertion/bulle/fusion) sur un tableau. Quelles sont les forces et les faiblesses de chacun?**

*void echanger (int\*a, int\* b)*

*{*

*int temporaire;*

*temporaire = \*a;*

*\*a = \*b;*

*\*b = temporaire;*

*}*

*tab[] = {‘P’, ‘L’, ‘O’, ‘B’, ‘V’, ‘X’, ‘B’};*

*Tri sélection: on trouve le plus petit, puis le deuxième plus petit et ainsi de suite. Pas beaucoup d’échange, pratique sur les listes chaînées.*

*void triSeletion (int\* tab, int n)*

*{*

*int min;*

*for (int i = 0; i < n – 1; i++)*

*{*

*min = i;*

*for (int j = i + 1; j < n; j++)*

*{*

*if (tab[min] > tab[j])*

*{*

*min = j;*

*}*

*}*

*echanger (&tab[i], &tab[min]);*

*}*

*}*

*Tri insertion : on prendre un élément à la fois et on l’insère au bon endroit dans le tableau. Efficace sur les petits tableaux, les tableaux presque en ordre, en ligne.*

*void triInsertion (int\* tab, int n)*

*{*

*for (int i = 1; i < n; i++)*

*{*

*for (int j = i; j > 0 && tab[j – 1] > tab [j]; j--)*

*{*

*echanger (&tab[j], &tab[j - 1];*

*}*

*}*

*}*

*Tri bulle : les petites éléments percolent, les gros éléments calent. On arrête quand rien n’a bougé. Efficace pour les tableaux presque en ordre, mais en moyenne très lent.*

*void triBulle (int\* tab, int n)*

*{*

*int temoin = 1;*

*int f = n;*

*while (temoin)*

*{*

*temoin = 0;*

*for (int i = 1; i < f; i++)*

*{*

*if (tab[i – 1] > tab[i])*

*{*

*echanger (&tab[i – 1], &tab[i]);*

*temoin = 1;*

*}*

*}*

*f--;*

*}*

*}*

*Tri fusion : fait encore moins d’opérations en utilisant un algorithme récursif. Le tri fusion est un algorithme récursif qui décompose une liste en deux sous-listes, trie chacune d’elle et les fusionne.*

*int triFusion (int tab, int n)*

*{*

*if (n <= 1)*

*{*

*return n;*

*}*

*int millieu = n / 2;*

*int\* tabM[milieu – 1];*

*int\* tabN[n – 1];*

*return tab;*

*}*

**Dans une fonction récursive, qu'est-ce que la condition de sortie, l'appel récursif et le calcul de plus? Transformez un algorithme récursif en algorithme itératif (non récursif).**

*La condition de sortie sert à arrêter la récursion, l’appel récursif sert à répéter le corps de la fonction et le calcul de plus c’est ce qui fait passer de la solution à la instance plus petite à la solution voulue.*

*Algorithme récursif :*

*void fonctionMystere (long n)*

*{*

*if (n == 0)*

*return;*

*fonctionMystere (n / 2);*

*printf ("%ld", n % 2);*

*}*

*Algorithme itératif (non récursi)f :*

*void fonctionIterative (long n)*

*{*

*if (n == 0)*

*return;*

*const long N = n;*

*int longueur = 1;*

*while (n / 2 > 0)*

*{*

*longueur++;*

*n = n / 2;*

*}*

*n = N;*

*int envers[longueur];*

*int i = 0;*

*while (n / 2 > 0)*

*{*

*envers[i] = n % 2;*

*n = n / 2;*

*i++;*

*}*

*envers[i] = n % 2;*

*for ( i = longueur – 1; i >= 0; i--)*

*printf("%d", envers[i]);*

*}*

**Qu'est-ce qu'un arbre, une racine, une feuille, un nœud? Qu'est-ce la profondeur? Qu'est-ce qu'un arbre binaire? De recherche? Appliquez le parcours en profondeur sur un arbre.**

*Un arbre est un ensemble de nœuds avec un nœud spécial appelé racine, chaque nœud a 0, 1 ou plusieurs enfants et chaque nœud a exactement un parent (sauf la racine, qui en a aucun).*

*La racine est un nœud spécial, avec aucun parent et 0, 1 ou plusieurs enfants qui est à la origine de l’arbre.*

*Une feuille est un nœud qui n’a pas d’enfants.*

*Un nœud est un élément de l’arbre (soit la racine, soit un enfant/parent).*

*Le parcours en profondeur est parcourir l’arbre la racine en premier, puis son enfant gauche, puis son enfant droit, récursivement.*

*Un arbre binaire est un arbre où chaque parent a 0, 1 ou 2 enfants.*

*Les arbres de recherche son des structures dynamiques optimisées pour la recherche, chaque nœud est entre son enfant gauche et son enfant droit.*

*void parcoursProfondeur(Noeud\* racine)*

*{*

*if (racine != NULL)*

*{*

*printf("%c", racine->donnee);*

*parcoursProfondeur (racine->g);*

*parcoursProfondeur(racine->d);*

*}*

*}*