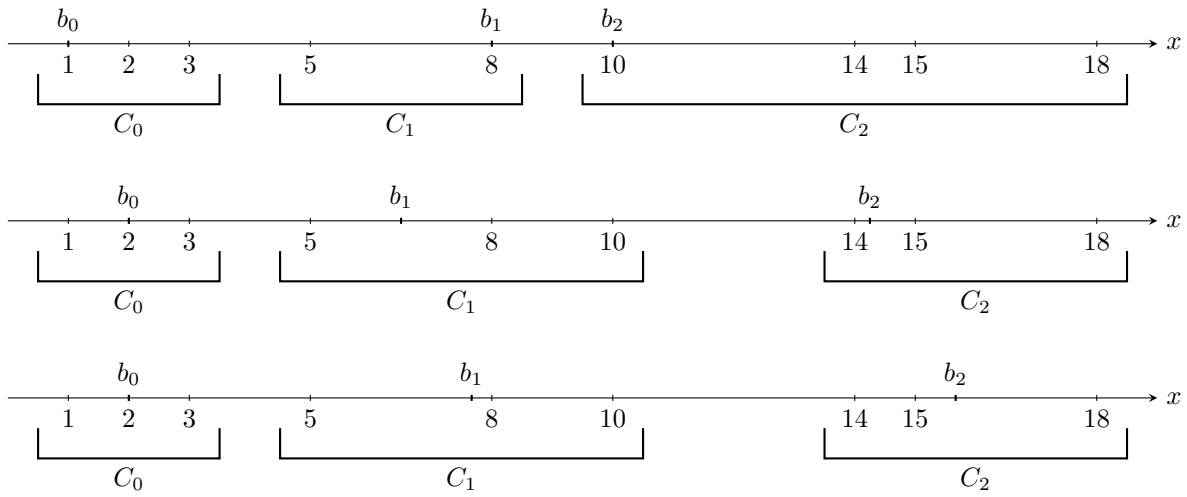
Clustering en dimension 1

1.1 Préliminaires

Question 1 Si $K = N$, chaque classe d'équivalence est de cardinal 1 (sinon l'une est vide) et le score est nul, ce qui est bien minimal. Si $K = N - 1$, toutes les classes sont de cardinal 1, sauf une qui est de cardinal 2 (par le principe des tiroirs). Pour minimiser le score, il faut mettre dans la même classe les deux éléments les plus proches de E.

Question 2 On a l'exécution suivante :



Question 3 Les erreurs sont :

- ligne 5 : il faut vérifier que ni `i1`, ni `i2` n'a atteint la dernière valeur. On doit remplacer la condition booléenne par `if (i2 == n2 || (i1 < n1 && tab1[i1] <= tab2[i2])){`
- entre les lignes 9 et 10 : il faut penser à incrémenter `i2` en rajoutant `i2++;`
- ligne 15 : le cas d'arrêt doit aussi prendre en compte le cas du tableau à un seul élément, sinon l'algorithme ne termine pas. Il faut remplacer cette ligne par `if (n <= 1) return;`
- entre les lignes 26 et 27 : il faut penser à libérer la mémoire. Il faut rajouter `free(tab1); free(tab2);`.

Question 4 Il suffit juste de faire attention aux indices.

```
double moyenne(double* E, int i, int j){
    double mu = 0;
    for (int k=i; k<j; k++){
        mu += E[k];
    }
    return mu / (j - i);
}
```

Question 5 Le code ressemble au précédent en faisant d'abord le calcul de la moyenne.

```
double somme_emc(double* E, int i, int j){
    double mu = moyenne(E, i, j);
    double S = 0;
    for (int k=i; k<j; k++){
        S += (E[k] - mu) * (E[k] - mu);
    }
    return S;
}
```

Question 6 Le tableau associé à la partition $\mathcal{P} = \{\{0,1,2\},\{3,4\},\{5,6,7\},\{8\}\}$ est `{0, 3, 5, 8}`. La partition associée au tableau `{0, 1, 4, 5}` est $\mathcal{P} = \{\{0\},\{1,2,3\},\{4\},\{5,6,7,8\}\}$.

Question 7 Il faut ici faire attention à la manière dont sont délimitées les classes d'équivalence. On traite la dernière classe à part, qui contient les valeurs jusqu'à x_{N-1}.

```
double score(double* E, int N, int* P, int K){
    double SP = somme_emc(E, P[K - 1], N);
    for (int i=0; i<K-1; i++){
        SP += somme_emc(E, P[i], P[i + 1]);
    }
    return SP;
}
```

Question 8 Il suffit d'une double boucle pour déterminer les indices des plus petits éléments de C_1 et C_2. On calcule le score à chaque étape et on garde la partition de score minimal. On fait attention à différencier l'usage d'un tableau statique et d'un tableau dynamique (pour pouvoir renvoyer le tableau, il faut l'allouer sur le tas).

```
int* clustering3(double* E, int N){
    int Pmin = malloc(3 * sizeof(int));
    Pmin[0] = 0; Pmin[1] = 1; Pmin[2] = 2;
    for (int i=1; i<N-1; i++){
        for (int j=i+1; j<N; j++){
            int P[3] = {0, i, j};
            if (score(E, N, P, 3) < score(E, N, Pmin, 3)){
                Pmin[1] = i; Pmin[2] = j;
            }
        }
    }
    return Pmin;
}
```

Question 9 On donne les complexités des fonctions précédentes :

- `moyenne(E, i, j)` est en $\mathcal{O}(j-i)$, au même titre que `somme_emc(E, i, j)` ;
- `score(E, N, P, K)` est en $\mathcal{O}(N)$, car on calcule `somme_emc` pour chaque classe, dont la somme des cardinaux est N (attention à ne pas faire une analyse trop grossière ici) ;
- on en déduit que `clustering3(E)` est en $\mathcal{O}(N^3)$, car on fait le calcul d'un score de l'ordre de $\mathcal{O}(N^2)$ fois. La création et copie des tableaux n'est pas à prendre en compte (car ce sont des tableaux de taille 3).

1.2 Clustering hiérarchique ascendant

Question 10 On garde en mémoire l'indice i_{opt} et l'écart de moyenne entre $C_{i_{\text{opt}}}$ et $C_{i_{\text{opt}}+1}$. Ensuite, on parcourt tous les indices jusqu'à $K-2$ et on vérifie si l'écart est strictement inférieur pour éventuellement mettre à jour. On fait toujours attention à traiter la dernière classe à part. C'est pour cette raison qu'il y a un opérateur ternaire. On rappelle que `b?x:y` renvoie `x` si `b` vaut `true` et `y` sinon.

```
int classes_plus_proches(double* E, int N, int* P, int K){
    int iopt = 0;
    double moy1 = moyenne(E, P[1], (K==2)?N:P[2]);
    int ecartopt = moy1 - moyenne(E, P[0], P[1]);
    for (int i=1; i<K-1; i++){
        double moy2 = moyenne(E, P[i + 1], (K==i+2)?N:P[i + 2]);
        double ecart = moy2 - moy1;
        if (ecart < ecartopt){
            iopt = i;
            ecartopt = ecart;
        }
        moy1 = moy2;
    }
    return iopt;
}
```

À noter, on aurait pu partir de la fin du tableau pour éviter les tests supplémentaires.

Question 11 On crée un nouveau tableau, qu'on remplit avec les éléments d'indice différent de `iopt + 1`.

```
int* fusion_classes(int* P, int K, int iopt){
    int* nouvP = malloc((K - 1) * sizeof(int));
    for (int i=0; i<K; i++){
        if (i <= iopt){
            nouvP[i] = P[i];
        } else if (i > iopt + 1){
            nouvP[i - 1] = P[i];
        }
    }
    return nouvP;
}
```

Question 12 On se contente d'appliquer l'algorithme décrit précédemment avec les fonctions déjà écrites. On pense à libérer la mémoire du tableau `P` lorsqu'on fusionne des classes.

```
int* CHA(double* E, int N, int K){
    int* P = malloc(N * sizeof(int));
    for (int i=0; i<N; i++) P[i] = i;
    int taille = N;
    while (taille > K){
        int iopt = classes_plus_proches(E, N, P, taille);
        int* nouvP = fusion_classes(P, taille, iopt);
        free(P);
        P = nouvP;
        taille--;
    }
    return P;
}
```

Question 13

– La fonction `classe_plus_proches` calcule de l'ordre de K moyennes, pour une somme des tranches d'indices égale à N. La complexité est donc en $\mathcal{O}(N)$.
– La fonction `fusion_classes` se contente de créer et remplir un tableau de taille $K-1$, donc en $O(K)$ (avec $K \leqslant N$).
– Finalement, la fonction `CHA` fait appel aux deux fonctions précédentes, pour $k \in [\![K, N]\!]$, soit une complexité en $O((N-K) \times N)$.

1.3 Solution optimale en programmation dynamique

Question 14 Lorsque $k = 1$, $D(n,k) = S_{\text{emc}}(0,n)$ (il n'y a qu'une seule classe). Ce score peut être calculé en $\mathcal{O}(n)$.

Question 15 Une partition des n éléments de taille k consiste en une partition de $i < n$ éléments en $k-1$ classes, à laquelle on rajoute une classe formée des $n-i$ derniers éléments. Comme aucune classe ne doit être vide, on considère $i \geqslant k-1$. La partition optimale atteint le minimum parmi toutes les partitions possibles de cette forme, ce qui donne bien la formule voulue.

Question 16 On écrit une fonction auxiliaire `cluster_rec` qui prend en argument l'ensemble E, des entiers n et k et un dictionnaire (ici codé par une matrice) mémoïsant les résultats, et renvoie $D(n,k)$. L'initialisation et l'hérédité se font selon les deux questions précédentes.

```
double cluster_rec(double* E, int n, int k, double** dic){
    if (dic[n][k] < 0){
        if (k == 1){
            dic[n][k] = somme_emc(E, 0, n);
        } else {
            dic[n][k] = cluster_rec(E, k - 1, k - 1, dic) + somme_emc(E, k - 1, n);;
            for (int i=k-1; i<n; i++){
                double d = cluster_rec(E, i, k - 1, dic) + somme_emc(E, i, n);
                if (d < dic[n][k]){
                    dic[n][k] = d;
                }
            }
        }
    }
    return dic[n][k];
}
```

Une fois cette fonction écrite, il suffit de lancer un appel avec $n = N$ et $k = K$, en utilisant un dictionnaire vide. On pense à libérer la mémoire avant de renvoyer la valeur.

```c
double clustering_dynamique(double* E, int N, int K){
    double** dic = malloc((N + 1) * sizeof(double*));
    for (int n=0; n<=N; n++){
        dic[n] = malloc((K + 1) * sizeof(double));
        for (int k=0; k<=K; k++){
            dic[n][k] = -1;
        }
    }
    double d = cluster_rec(E, N, K, dic);
    for (int n=0; n<=N; n++){
        free(dic[n]);
    }
    free(dic);
    return d;
}
```

Question 17 On remarque qu'il y a de l'ordre de $N \times K$ valeurs qui sont calculées dans le dictionnaire. De plus, chaque valeur nécessaire de calculer le minimum par la boucle `for`. Cette boucle, de taille $n - k$, fait un appel à `somme_emc`, de complexité $\mathcal{O}(n - i)$. En combinant tout ça, on obtient une complexité totale en $\mathcal{O}(K \times N^3)$.

Question 18 Dans le calcul du minimum, on peut garder en mémoire l'indice `i` qui permet d'atteindre ce minimum, ce qui correspond au plus petit élément de la classe C_{k-1} dans une partition de taille k. On peut alors reconstruire une solution complète en utilisant les valeurs présentes dans le dictionnaire.