Aufgabe 1. Gegeben seien n Listen, die jeweils m Zahlen in aufsteigender Reihenfolge enthalten. Betrachten Sie den folgenden Lösungsalgorithmus, der daraus eine einzelne, geordnete Liste von $N = nm$ Zahlen erzeugt: Zunächst wird die erste Liste mit der zweiten so verschmolzen, dass eine geordnete Liste der Länge $2m$ entsteht. Danach wird die so erhaltene Liste mit der dritten Liste verschmolzen, sodass nun eine geordnete Liste mit $3m$ Elementen entsteht. Dieses Prozedere wird mit jeder weiteren noch nicht verschmolzenen Liste wiederholt bis man eine Liste der Länge nm erhält, die dann zurückgegeben wird.

(a) Berechnen Sie die Worst-Case Laufzeit des oben skizzierten Algorithmus.

(b) Entwerfen Sie einen effizienteren Algorithmus, der einen Heap beim Vergleich von Elementen aus verschiedenen Listen verwendet. Geben Sie die Worst-Case Laufzeit von dem entworfenen Algorithmus an.

---

a)
Zwei sortierte Listen verschmelzen: O(x) (x ... Gesamtzahl der Elemente in den Listen)

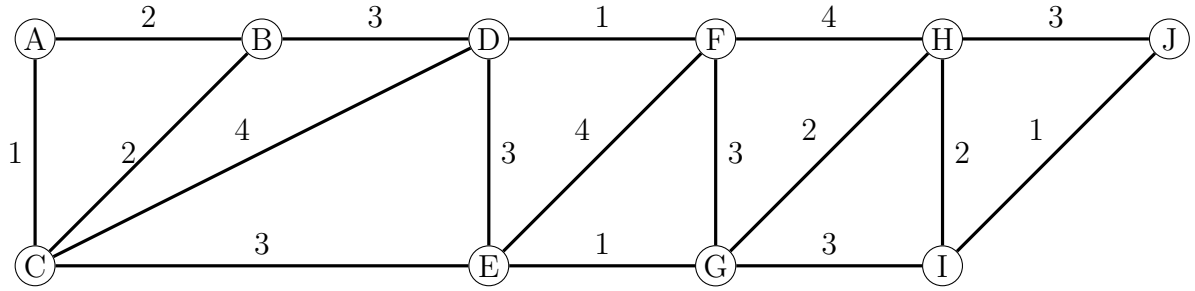
Erste Liste mit der zweiten verschmelzen: O(2m)
Erhaltene mit der dritten verschmelzen: O(3m)
Erhaltene mit der vierten verschmelzen: O(4m)
...
Erhaltene mit der nten verschmelten: O(nm)

Laufzeit: O(2m+3m+...+nm) = O(m*(2+3+...+n) = O(m*((n+2)*(n-1)/2))
= O(m*(n^2+n-2)/2) = O(mn^2/2 + mn/2 - 1) = O(mn^2)

b)
Erstelle einen Min-Heap aus allen Listen. Danach immer das kleinste Element entfernen und zu einer Liste hinzufügen bis der Heap leer ist.

Erstellen des Min-Heaps: O(n*m)
Kleinstes Element entfernen: O(log(n*m))
Kleinstes Element entfernen für alle Elemente durchführen: O(nm*log(nm))
Summe: O(nm*log(nm))

Aufgabe 2. Gegeben sei der nachfolgend abgebildete ungerichtete, kantengewichtete Graph mit 10 Knoten.



(a) Verwenden Sie den Algorithmus von Kruskal, um einen minimalen Spannbaum für diesen Graphen zu finden. Notieren Sie die Reihenfolge, in der Sie die Kanten hinzufügen.

(b) Verwenden Sie den Algorithmus von Prim, um einen minimalen Spannbaum für diesen Graphen zu finden. Beginnen Sie mit Knoten A und notieren Sie die Reihenfolge, in der Sie die Kanten hinzufügen.

(c) Vergleichen Sie die Ergebnisse der beiden Algorithmen. Sind die resultierenden minimalen Spannbäume identisch? Kann es korrekte Ausführungen beider Algorithmen auf der obigen Probleminstanz geben, sodass die zurückgegebenen minimalen Spannbäume übereinstimmen?

---

a/b) Siehe Ende
c)
Die MST sind identisch. Dies zeigt dass es korrekte Ausführungen beider Algorithmen auf der obigen Probleminstanz geben kann, sodass die zurückgegebenen minimalen Spannbäume übereinstimmen.

Aufgabe 3. Sei $G = (V, E)$ ein Graph mit nicht-negativen Kantengewichten. Gegeben sind ein MST T von G, sowie ein kürzester Pfad P von s nach t in G mit $s, t \in V$. Sei G' der gleiche Graph wie G mit dem Unterschied, dass das Gewicht jeder Kante in E nun um 1 höher ist. Beweisen oder widerlegen Sie (z.B. mit Gegenbeispiel) folgende Aussagen:

(a) T ist auch ein MST von G'.

(b) P ist auch ein kürzester Pfad von s nach t in G'.

---

a)
Der Algorithmus von Prim (oder auch der von Kruskal) würde immer noch die exakt selben Kanten auswählen, da wenn man zu allen Elementen einer Menge 1 dazuzählt sich weder die Sortierreihenfolge oder das Minimum ändern würden.
Da beide Algorithmen garantiert einen MST produzieren (den selben MST T wie auch im ersten Fall) ist T auch ein MST von G'.

b)
Wenn s und t über drei Kanten mit jeweils Gewicht 1 (Summe 3), sowie eine einzelne Kante mit Gewicht 4 verbunden wären wäre der kürzeste Pfad der über die drei Kanten.
Wenn nun alle Gewichte um 1 erhöht werden würden wären s und t über drei Kanten mit jeweils Gewicht 2 (Summe 6) und eine einzelne mit Gewicht 5 verbunden. Der kürzeste Pfad in G' wäre nun der über die einzelne Kante und damit ein anderer wie in G.

Aufgabe 4. Gegeben sind n Projekte $p_1, \ldots, p_n$. Für $1 \leq i \leq n$ bezeichnet s_i die Kosten für das Projekt p_i. Nehmen Sie an, es liegt ein Budget $B \in \mathbb{N}$ mit $B < \sum_{i=1}^{n} s_i$ vor.

(a) Maximiert ein Greedy-Algorithmus, der die Projekte nach *aufsteigenden* Kosten unter Berücksichtigung von B auswählt, die Anzahl der ausgewählten Projekte? Geben Sie einen Beweis oder ein Gegenbeispiel an.

(b) Maximiert ein Greedy-Algorithmus, der die Projekte nach *absteigenden* Kosten unter Berücksichtigung von B auswählt, die Auslastung des Budgets? Geben Sie einen Beweis oder ein Gegenbeispiel an.

---

a)
Ja, da immer das Projekt mit den geringsten Kosten zuerst ausgewählt wird
und viele kleine Projekte die Anzahl maximieren.

b)
Gegenbeispiel: Budget=100, drei Projekte mit Kosten 98, 50 und 49.
Dann würde der Algorithmus 98 (Auslastung 98/100) auswählen, wodurch einerseits
keine weiteren Projekte mehr ausgewählt werden könnten und auch das Budget
weniger gut ausgelastet ist als wenn die Projekte mit Kosten 50 und 49 (99/100)
ausgewählt werden würden.

Aufgabe 5. Betrachten Sie *Meanquicksort*, eine Variante von Quicksort zum Sortieren von Zahlen, die immer das arithmetische Mittel der jeweils zu sortierenden Subfolge als Pivot-Element verwendet. Ist das arithmetische Mittel kein Element der Subfolge, dann erfolgt das Aufteilen trotzdem durch Vergleich mit dem arithmetischen Mittel.

(a) Führen Sie Meanquicksort auf das folgende Array aus und implementieren Sie den Schritt des Aufteilens so, dass die Elemente einer Subfolge immer in der gleichen Reihenfolge angeordnet werden wie in der Originalfolge. Geben Sie die Zwischenschritte wie im Foliensatz „Divide-and-Conquer“ an.

$$[7, 3, 18, 0, 5, 9, 11]$$

(b) Erklären Sie, wie Sie für ein beliebiges n eine Beispielsequenz der Länge n erstellen können, die zeigt, dass die Worst-Case Laufzeit von Meanquicksort auch nicht besser als quadratisch sein kann. Die Beispielsequenz darf dabei keine Duplikate enthalten.

Hinweis: Es reicht nicht, einzelne Instanzen fester Größe als Beispielinstanzen anzugeben.

```
a)
[7, 3, 18, 0, 5, 9, 11] Mittel: 7,57
[7, 3, 0, 5][18, 9, 11] Mittel: 3,75
[3, 0][7, 5][18, 9, 11] Mittel: 1,50
[0][3][7, 5][18, 9, 11] Mittel: 6,00
[0][3][5][7][9, 11][18] Mittel: 12,66
[0][3][5][7][9][11][18] Mittel: 10

[0, 3, 5, 7, 9, 11, 18] Done

b)
Fixed size example: [a, 3a, 8a, 24a]
(a+3a+8a+24a)/4 = 9a => Liste wird zwischen 8a und 24a aufgeteilt =>
Verhalten Äquivalent zu letztes Element ist Pivot.
(a+3a+8a)/3 = 4a => Liste wird zwischen 3a und 8a aufgeteilt =>
Verhalten Äquivalent zu letztes Element ist Pivot.
(a+3a)/2 = 2a => Liste wird zwischen a und 3a aufgeteilt =>
Verhalten Äquivalent zu letztes Element ist Pivot.

Variable size example (i ... index (Starting with 0), sp ... sum of all previous elements):
[a, 4*3^(i-2)*a-sp, 4*3^(i-2)*a-sp, ..., 4*3^(i-2)*a-sp]

Das letzte Element ist immer pivot => Laufzeit O(n²) (siehe Slides)
```

Algorithmus

Mengen	Kante	Hinzu?	T
{A}, {B}, {C}, {D}, {E}, {F}, {G}, {H}, {I}, {J}	(A,C)	Ja	{(A,C)}
{A, C}, {B}, {D, F}, {E}, {G}, {H}, {I}, {J}	(D,F)	Ja	{(A,C), (D,F)}
{A, C}, {B}, {D, F}, {E, G}, {H}, {I}, {J}	(E,G)	Ja	{(A,C), (D,F), (E,G)}
{A, C}, {B}, {D, F}, {E, G}, {H}, {I, J}	(I,J)	Ja	{(A,C), (D,F), (E,G), (I,J)}
{A, B, C}, {D, F}, {E, G}, {H}, {I, J}	(A,B)	Ja	{(A,C), (D,F), (E,G), (I,J), (A,B)}
{A, B, C}, {D, F}, {E, G}, {H}, {I, J}	(B,C)	Nein	{(A,C), (D,F), (E,G), (I,J), (A,B)}
{A, B, C}, {D, F}, {E, G, H}, {I, J}	(G,H)	Ja	{(A,C), (D,F), (E,G), (I,J), (A,B), (G,H)}
{A, B, C}, {D, F}, {E, G, H, I, J}	(H,I)	Ja	{(A,C), (D,F), (E,G), (I,J), (A,B), (G,H), (H,I)}
{A, B, C, D, F}, {E, G, H, I, J}	(B,D)	Ja	{(A,C), (D,F), (E,G), (I,J), (A,B), (G,H), (H,I), (B,D)}
{A, B, C, D, F, E, G, H, I, J}	(C,E)	Ja	{(A,C), (D,F), (E,G), (I,J), (A,B), (G,H), (H,I), (B,D), (C,E)}

Ausgewählt	Knotenmenge S	Kantenmenge T
A	A	
C	A,C	(A,C)
B	A,C,B	(A,C),(A,B)
D	A,C,B,D	(A,C),(A,B),(B,D)
F	A,C,B,D,F	(A,C),(A,B),(B,D),(D,F)
E	A,C,B,D,F,E	(A,C),(A,B),(B,D),(D,F),(C,E)
G	A,C,B,D,F,E,G	(A,C),(A,B),(B,D),(D,F),(C,E),(E,G)
H	A,C,B,D,F,E,G,H	(A,C),(A,B),(B,D),(D,F),(C,E),(E,G),(G,H)
I	A,C,B,D,F,E,G,H,I	(A,C),(A,B),(B,D),(D,F),(C,E),(E,G),(G,H),(H,I)
J	A,C,B,D,F,E,G,H,I,J	(A,C),(A,B),(B,D),(D,F),(C,E),(E,G),(G,H),(H,I),(I,J)