1. Ex 24.3.6

Define an array of pointers \( P = [P_0, P_1, P_2, \ldots, P_n, P_{n+1}] \)
where each pointer \( P_i \) is the head pointer which points to a linked list, see fig below.

Each linked list is a doubly linked, circular list, and the linked list \( P_i \) points to contains all the nodes in the graph whose \( d[X] = i \).

Because \( E \rightarrow \{0, 1, \ldots, W\} \), Therefore the largest possible \( d[X] \) will be \( W(|V| - 1) \). Thus to build the pointer array, we need \( W(|V| - 1) \) pointers, plus the last one \( P_0 \) which denotes the linked list containing nodes whose \( d = \infty \).

Therefore, under such priorityqueue, INSERT can be done in \( O(1) \) time, DELET DECREASE-KEY can be done in \( O(1) \) time, \( \ldots \), the INITIALIZE-SINGLE-SOURCE costs \( O(V) \) times, and the total RELAX operations cost \( O(E) \) times.

We now look at Extract-Min, consider a sequence of \( |V| \) Extract-Min after performing \( |V| \) insertions the Dijkstra classes, since there are at most \( |V| \) insertions,
and the list is key for each pointer is increasing, therefore we use min pointers to keep track of the location of index after each Extract-Min, therefore, to run Extract-Min at second time can just start from the min pointer, points to.

Therefore, the sequence of \( IVI \) Extract-Min costs \( O(IVI + W(IVI - I)) \) time.

The total time for the modified Dijkstra algo becomes:
\[
O(IVI + IEI + IVI + WIVI - W) \\
= O(WV + E)
\]

2. Problem 24-1. Yen's improvement to Bellman-Ford
(a) Proof. According to the definition of \( G_f \):
\[
G_f = \{(V_i, V_j) \mid E = i < j \} \text{ edges are ordered from small index to large one, therefore } G_f \text{ is topologically sorted,}
\]

(b) Assume \( G_f \) is cyclic, thus there's a cycle in \( G_f \), which means there must be some \( (V_i, V_j) \) where \( i > j \), this contradicts the fact that \( i < j \).

\( G_f \) is acyclic.

From (a) and (b), \( G_f \) is acyclic and topologically sorted.

Same proof with \( G_b \).
(c) From part (b), we know only \( \lceil \sqrt{|V|} \rceil \) passes over edges. Therefore, the new running time becomes
\[
O(\frac{\sqrt{|V|}}{2} \cdot E) = O(VE)
\]
1. Equals the previous running time.


(a) "Transitive closure can be represented by a boolean matrix, therefore, let
\[
TC[i, j] = \begin{cases} 
1 & \text{if there is a path from } i \text{ to } j \text{ for } 1 \leq i, j \leq |V| \\
0 & \text{otherwise}
\end{cases}
\]

The update algo: (Assume we add an edge \((u, v)\),
1. for \( i = 1, 2, \ldots, |V| \)
2. for \( j = 1, 2, \ldots, |V| \)
   - if \( TC[i, u] = TC[v, j] = 1 \)
     - then  \( TC[i, j] = 1 \)

The running time is thus \( O(|V|^3) \)
7. Ex 5.2-4.

Let $X_i = \begin{cases} 1 & \text{if the } i\text{th customer gets back his hat} \\ 0 & \text{otherwise} \end{cases}$

:. $P(X_i = 1) = \frac{1}{n}$< Why? 

:. $E[X] = E\left[\sum_{i=1}^{n} X_i\right]$ Because the hat-check person gives the hats back to the customers in a random order. There are $n!$ permutations for the return order, $(n-1)!$ of them have the entry equal to $i$. 

:. $= \frac{\sum_{i=1}^{n} 1}{n}$ 

:. $= \frac{n}{n} = 1$ 

:. 1 customer is expected to get his hat back.

8. Ex 7.4-4 According to Page 158,

$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$

:. $= \sum_{i=1}^{n-1} \frac{n!}{k+1} > \sum_{i=1}^{n-1} \frac{1}{k+1} = \sum_{i=1}^{n-1} \frac{1}{i+1}$

:. $E[X]$ is $\Omega(n \log n)$. 

Bolgia*
9. Ex 7.4-5.

1. The algorithm starts by running quick sort; it keeps on running until the number of each subproblem becomes $k$, then apply insertion sort. The elements in

Therefore, for the number of "final" subproblems is \( \frac{n}{k} \) solving each subproblem with insertion sort cost \( O(k^2) \).

\( \therefore \) totally \( \frac{n}{k} \times O(k^2) = O(nk) \)

\( \Rightarrow \) divide progress.

For the quicksort (divide-and-progress):

the height of the tree is \( \lg n - \lg k = \lg \frac{n}{k} \)

For each level of the tree, the total cost is \( O(n) \).

\( \therefore \) totally \( \lg \frac{n}{k} \cdot O(n) = O(n \cdot \lg \frac{n}{k}) \)

\( \therefore \) Running time is \( O(nk + n \cdot \lg \frac{n}{k}) \).

2. Let \( f(k) = nk + n \cdot \lg \frac{n}{k} \), to choose \( k \) for optimization, we calculate \( f'(k) = 0 \).

\( k = \sqrt[2]{2n} \) (Theoretically)

Practically, depends on the array's property. For example, if the array is almost sorted, \( k \) should be as large as possible.

To compute maximal value of function \( f(k) \), one approach is to use the let the differential \( f'(k) = 0 \), solve the constraint obtaining \( k \), which makes \( f(k) \) maximal.
Ex. C.3-3  Bet $1 on any number 1 to 6. Roll 3 dice.

If chosen number doesn't appear, lose $1.
If " " appears once, win $1.
If " " appears twice, win $2.
If " " appears three times, win $3.

What is expected gain? Fix chosen number to be i.
Let X = amount won (or lost).

\[ E[X] = \sum_{v=-1,1}^2 v \cdot \Pr[X = v] \]

Need to calculate \( \Pr[X = -1] \), ..., \( \Pr[X = 3] \).

\( \Pr[X = -1] = \Pr[\text{i doesn't appear}] = \frac{1}{6} \cdot \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{216} \)

\( \Pr[X = 1] = \Pr[\text{exactly 1 die rolls i}] = \frac{1}{6} \cdot \frac{5}{6} \cdot \frac{5}{6} \cdot 3 = \frac{75}{216} \)

\( \Pr[X = 2] = \Pr[\text{exactly 1 die does not roll i}] = \frac{1}{6} \cdot \frac{1}{6} \cdot \frac{5}{6} \cdot 3 = \frac{15}{216} \)

\( \Pr[X = 3] = \Pr[\text{all 3 dice roll i}] = \frac{1}{6} \cdot \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{216} \)

So \( E[X] = (-1) \cdot \frac{125}{216} + 1 \cdot \frac{75}{216} + 2 \cdot \frac{15}{216} + 3 \cdot \frac{1}{216} \)

\( = \frac{-14}{216} \)

(expect to lose money)
Ex. C-2-3: Deck of 10 cards, numbered 1 to 10, is shuffled.

Three cards are removed one at a time. What is the probability that the 3 cards are selected in increasing order?

\[
\Pr \left( 1^{st} \leq 2^{nd} \text{ and } 2^{nd} < 3^{rd} \right) = \sum_{\text{all subsets } \{a, b, c\} \text{ of the cards}} \frac{1}{\binom{10}{3}} \cdot \frac{1}{3!}
\]

There are 3! different permutations of \(\{a, b, c\}\), all are equally likely, and only one is in sorted order.

\[
= \binom{10}{3} \cdot \frac{1}{3!} = \frac{1}{6}
\]

Ex. C.3-2: A set \(\{1 \ldots n\}\) of \(n\) distinct numbers is randomly ordered, with each permutation equally likely. What is expectation of index of maximum element in array?

Let \(X = \text{index of max. index}\).

\[
\mathbb{E}[X] = \sum_{\nu = 1}^{\nu} \nu \cdot \Pr [X = \nu]
\]

\[
= \frac{1}{n} \cdot \sum_{\nu = 1}^{n} \nu = \frac{1}{n} \cdot \frac{n(n+1)}{2}
\]

\[
= \frac{n+1}{2}
\]