1. Have an output buffer of size $BD$ for each value in the range $[1, B]$. Bring $BD$ elements at a time from the disks into the main memory. Distribute these keys to the buffers based on the key values. Repeat this process. When any buffer is full, write these $BD$ elements into the disks. In the disks, we will grow $B$ runs in separate regions. After one read pass through the data, $X$ has been sorted into $B$ runs in the disks.

Now we have to write the runs contiguously in the disks. This can be done in one more pass through the data.

2. If $s$ is a repeated substring of $X$, it must be a prefix for at least two different suffixes. Also, $s$ must be the path label of an internal node. Thus, to find the longest repeated substring of $X$, we can construct a suffix tree $Q$ for $X$ and look for an internal node $u$ in $Q$ of the largest string depth. We then output the path label of $u$. The total time is $O(n)$.

3. Let $P$ be the pattern whose length is $n$. Note that there are only $(\sigma - 1)n$ strings $P'$ of length $n$ such that the Hamming distance between $P$ and $P'$ is 1. Each such $P'$ is called a 1-neighbor of $P$. For example, let $\Sigma = \{a, b, c\}$ and $P = bcab$, then the only strings $P'$ of length 4 such that the Hamming distance between $P$ and $P'$ is 1 will be acab, ccab, baab, bbab, bcbb, bccbb, bcaa, and bcac.

We first construct a suffix tree $Q$ on $T$ in $O(m)$ time. Followed by this we generate all the 1-neighbors of $P$. This can be done in $O(\sigma n^2)$ time. For every 1-neighbor $P'$ of $P$ we use $Q$ to identify all the occurrences of $P'$ in $T$. We can also find all the occurrences of $P$ in $T$. For each $P'$ we spend $O(n)$ time. For $P$ also we spend $O(n)$ time.

Thus the total run time of the algorithm is $O(m + \sigma n^2)$.

4. Let $T$ be the text and $P$ be the pattern with $|T| = m$ and $|P| = n$. Let $S[1 : m]$ be the suffix array for $T$. Specifically, $S[i]$ specifies the starting position of the $i^{th}$ smallest suffix of $T$, for $1 \leq i \leq m$. As was shown in the homework problem (HW2, Problem 5), given two suffixes $T_i$ and $T_j$ (with $T_i < T_j$), we can check if $P$ falls in the interval $[T_i, T_j]$ in $O(1)$ time using $n$ common CRCW PRAM processors.

We partition the interval $[1, m]$ into $\sqrt{m}$ equal subintervals: $[1, \sqrt{m}], [\sqrt{m} + 1, 2\sqrt{m}]$, etc. Assign $n$ processors per subinterval. Let $[i, j]$ be any of these subintervals. The $n$ processors associated with this interval check if $P$ lies in the interval $[T_{S[i]}, T_{S[j]}]$. This is done in parallel for each subinterval. Clearly, this step takes $O(1)$ time. There are two cases to consider:

**Case 1:** $T_{S[i]} = P$. In this case, there will be some number of consecutive subintervals such that the entire interval corresponds to matches and in the next subinterval $[i, j]$, there is a $k$ such that $P \neq T_{S[k]}$. We have to find the least $k \in [i, j]$ such that $P \neq T_{S[k]}$. This can be done in $O(1)$ time by assigning $n$ processors for each $k$ in this subinterval.
Case 2: $T_3[1] \neq P$. In this case the matches will start in a subinterval, span some number of subintervals and end in a subinterval. In this case we can find the starting index and the ending index for the matches in a similar manner.