חלק יבש :

נשמור את המבנה בעצים ובמצביעים הבאים:

1) **System :**
מבנה נתונים המכיל את הסטטיסטיקות של כל חברה שבמערכת ועל כלל עובדיה, אשר מוצגים בעזרת העצים הבאים:
1. **Companies**- עץ AVL שמטרתו לשמור את כל החברות שבמערכת – מסודרים לפי Company_ID, כאשר ה- data של כל חברה המופיעה בעץ הוא שווי החברה.
2. **SettledCompanies –** עץ AVL שמטרתו לשמור את כל החברות בעלות עובדים – מסודרים לפי **Company_ID** , כאשר ה-data מכיל את כל הפרטים על החברה מטיפוס Company.
3. **SalarySortedEmp** – עץ AVL שמטרתו לשמור את כלל העובדים שבמערכת – מסודרים לפי המשכורת שלהם.
4. **IdSortedEmp**- עץ AVL שמטרתו לשמור את כלל העובדים שבמערכת – מסודרים על פי המספר המזהה שלהם.

2) **Company:**
מבנה נתונים המכיל את פרטי החברה:
1. **Id** – מספר מזהה של החברה.
2. **Value** – שווי החברה.
3. **salarySorted** – עץ AVL שמטרתו לשמור את כלל העובדים בחברה – מסודרים לפי המשכורת שלהם.
4. **IdSortedEmp** - עץ AVL שמטרתו לשמור את כלל העובדים בחברה – מסודרים על פי המספר המזהה שלהם.

3) **Employee**:
מבנה נתונים המכיל פרטים על עובד:
1. **Id**- מספר מזהה של העובד.
2. **Salary** – משכורת של העובד.
3. **Grade** – דרגה של העובד.
4. **CompanyID** – מצביע מסוג int אשר מצביע על מספר המזהה של החברה בה העובד עובד.

4) **Tnode**:
אובייקט (צומת) בעץ אשר מכיל בתוכו:
1. **Key –** מבציע מטיפוס KeyType שמייצג את המפתח של האובייקט.
2. **Data –** מצביע מטיפוס DataType שמייצג את המידע של האובייקט.
3. **Left** – מצביע לילד השמאלי.
4. **Right** – מצביע לילד הימני.
5. **Parent** – מצביע להורה של האובייקט.
6. **Height** – גובה של הצומת, מחושב על פי המרחק מהצומת אל העלה העמוק ביותר בתת העץ כאשר הצומת היא השורש.

5) **Tree** – עץ AVL :
1. **Root** – מצביע לאובייקט שעץ שהוא שורש של העץ.
2. **Counter** – מונה שתפקידו למנות כמה אובייקטים יש בעץ בכל זמן נתון.
3. **Max_node** - מצביע לאובייקט בעל הkey הגדול ביותר בעץ בכל זמן נתון.

סימונים: n – מספר העובדים במערכת. k – מספר החברות במערכת.

עץ AVL ופונקציות עזר:

מימשנו עץ AVL כפי שנלמד בהרצאות ובתרגולים, בנוסף מימשנו את הפונקציות עזר הבאות :

1) mergeTree – פונקציה המקבלת עץ AVL ,לוקחת את העץ ממנו קראו לפונקציה , ומחזירה עץ AVL משותף . נסמן את m להיות מספר האובייקטים בעץ הראשון ואת d להיות מספר האובייקטים בעץ השני. הפעולות שביצענו:
   1. הפיכת עץ באובייקט למערך ממוין באמצעות סיור InOrder. (בסה"כ 2 מערכים של ה- KeyType של כל אחד מהעצים ועוד 2 מערכים עבור כל ה- DataType).
   2. חיבור המערכים הממוינים המתאימים למערך אחד באמצעות Merge. (נקבל 2 מערכים בגודל d+m , אחד ממוין לפי KeyType ובשני DataType בהסתמך על המיון של KeyType).
   3. מחיקת כל האובייקטים מהעצים באובייקט ממנו נקראה הפונקציה.
   4. קריאה לפונקציה הרקורסיבית arrayToTree שמקבלת המערכים הסופיים, אינדקסים של התחלה וסיום, ומצביע לסטטוס הפעולה, והופכת אותו לעץ:
      1. הכנסת Array[middle] לשורש.
      2. קריאה לרקורסיבית לתת העץ השמאלי.
      3. קריאה רקורסיבית עבור התת עץ הימני.

סיבוכיות הריצה : של mergeTree היא : $O(d+m)$, מתבטא במספר הקריאות הרקורסיביות arrayToTree.

סיבוכיות המקום: $O(d+m+\log(d+m))$ **,** הקצאה של מערכים בגודל d+m וקריאה לarrayToTree.

findMaxNode – פונקציה רקורסיבית שמקבלת אובייקט (node) ובודקת אם הערך של האובייקט גדול יותר מהערך של max_node, אם כן ,מעדכנת את max_nude ומבצעת קריאה רקורסיבית עם תת העץ הימני. מבחינת **סיבוכיות ריצה והמקום** : $O(logm)$ כאשר $m = number\ of\ objects\ in\ the\ tree$.

2) **countCondNodes** -פונקציה רקורסיבית שמקבלת אובייקט מתוך העץ שהוא בעצם השורש של תת העץ , ומקבלת את הטווח המינימלי והטווח המקסימלי מטיפוס KeyType ומחזירה את מספר האובייקטים שנמצאים בטווח . **סיבוכיות הזמן** : $O(employees\ in\ range)$**.** שבמקרה הגרוע ביותר הוא שכל העובדים של החברה נמצאים בטווח : $O(n_{company})$. **סיבוכיות מקום:** $O(log(employees\ in\ range))$.

3) **specialcountCondNodes** - פונקציה רקורסיבית שמקבלת מערך של מצביעים לטיפוס מסוג DataType, ובנוסף מקבלת אובייקטים מסוג KeyType שמייצגים את הטווח המינימלי והמקסימלי, הפונקציה מעדכנת את מערך המצביעים במצביעים לטיפוס DataType שה-keyType שלו עומד בתנאי הטווח, פונקציה זאת תעזור לנו לממש פעולות במתודות עתידיות . **סיבוכיות זמן ריצה:** $O(employees\ in\ range)$**.**
סיבוכיות מקום: $O(log(employees\ in\ range))$.

4) **deleteAllNodes** - פונקציה רקורסיבית אשר מוחקת את כל האובייקטים בעץ , ומספר הפעמים שהפונקציה תיקרא היא כמספר האובייקטים בעץ. ולכן **סיבוכיות הריצה :** $O(m)$**, וסיבוכיות מקום :** $O(logm)$ **,** כאשר: $m = number\ of\ objects\ in\ the\ tree$**.**

5) ***arrayToTree*** – פונקציה רקורסיבית שמקבלת מערך מצביעים לאוביקטים מסוג *KeyType* ומערך מצביעים לאוביקטים מסוג *DataType* שמהם בונה עץ *AVL* שממוין לפי ה*KeyType* . הפונקציה מחזירה מצביע לאובייקט חדש בעץ , כאשר בסיום הרקורסיה נוצר עץ *AVL* .
סיבוכיות הזמן : $O(m) - m = number\ of\ objects\ in\ the\ tree$ **.**
סיבוכיות מקום: $O(logm) - m = number\ of\ objects\ in\ the\ tree$ **.**

פירוט הפונקציות בSystem:

1) void* Init() -
אתחול של המערכת System – כאשר בתוכה 4 עצי AVL ריקים שמצביעים ל- NULL .
סיבוכיות זמן : $O(1)$, לא מדובר על פונקציה רקורסיבית או איטרטיבית.
סיבוכיות מקום: O(1) , המערכת ריקה.

2) StatusType AddCompany(void *DS, int CompanyID, int Value) -
פונקציה שמכניסה חברה חדשה למערכת.
1. אם נשלח מצביע לא תקין של המערכת או מזהה חברה אי חיובי נזרקות שגיאות.
2. בדיקה האם החברה כבר קיימת במערכת או לא , אם כן תזרק שגיאה $O(logk)$.
3. יצירת אובייקט מטיפוס Company עם המזהה שהתקבל ויצירת העצים והשדות של האובייקט $O(1)$.
4. הכנסת החברה לעץ Companies במערכת :
   1. מציאת המקום המתאים בעץ $O(logk)$.
   2. הכנסת האובייקט לעץ - $O(1)$.
   3. ביצוע גלגול אם צריך - $O(1)$.

הפונקציה תחזיר SUCCESS במקרה וכל התהליך הסתיימו בצורה תקינה , אחרת תיזרק שגיאת זיכרון.

סיבוכיות זמן : $O(logk)$.
סיבוכיות מקום: $O(logk)$ מכיוון שאופן החיפוש הוא רקורסיבי ,החברה מאותחלת והיא ריקה.

3) StatusType AddEmployee(void *DS, int EmployeeID, int CompanyID, int Salary, int Grade)
פונקציה שמכניסה עובד חדש לחברה.
1. אם נשלח מצביע לא תקין של המערכת או מזהה עובד / מזהה חברה / משכורת אי חיוביים, נזרקות שגיאות.
2. בדיקה אם החברה לא קיימת במערכת, אם כן נזרקת שגיאה – O(logk).
3. בדיקה אם קיימים עובדים במערכת, אם לא נבצע את הפעולות הבאות:
   1. נקצה עץ עובדים למערכת – O(1). תיזרק שגיאה במקרה וההקצאה נכשלה.
   2. נקצה עץ משכורות למערכת – O(1) . תיזרק שגיאה במקרה וההקצאה נכשלה.
4. בדיקה אם שחקן נמצא במערכת , מתבצע על ידי מעבר בעץ IdSortedEmp , אם כן נזרקת שגיאה – O(logn) .
5. חיפוש בעץ SettledCompanies ויצירת מצביע לData – O(logk) ,אם החברה לא קיימת , אז מדובר על עובד ראשון ,אז נבצע את הפעולות הבאות:
   1. ניצור מבציע לחברה ע"י חיפוש בעץ companies - O(logk).
   2. ניצור אובייקט מטיפוס Company -O(1), נזרוק שגיאת זיכרון במקרה ויש תקלה.
   3. נקצה עץ עובדים לחברה ועץ משכורות לחברה - $O(1)$.
   4. נכניס את החברה לעץ בעזרת insert -O(logk) , במקרה ולא נזרוק שגיאה, ונקשר למצביעים המתאימים את עצי העובדים והמשכורות.
6. ניצור אובייקט Employee – O(1).
7. נכניס את ה-Employee ל-2 העצי החברה - $O(logm) + O(logm)$, כאשר m הוא מספר העובדים בחברה. מכיוון ש $m \leq n$ אז $O(logm) \leq O(logn)$.
8. נכניס ל-2 עצי העובדים במערכת - $O(logn) + O(logn)$

הפונקציה תחזיר SUCCESS במקרה וכל התהליך הסתיימו בצורה תקינה , אחרת תיזרק שגיאת זיכרון.

סיבוכיות זמן :
$$O(logk) + O(1) + O(1) + O(logn) + O(logk) + O(logk) + O(1) + O(1) + O(logk) + O(1) + O(logm) + O(logm) + O(logn) + O(logn) = 5logk + 3logn + 2logm + 5 = \boldsymbol{O(logk + logn)}$$

סיבוכיות מקום: $O(logn + logk)$ רקורסיבית .

4) StatusType RemoveEmployee(void *DS, int EmployeeID) –
הפונקציה מוחקת את נתוני העובד מהמערכת.

1. אם נשלח מצביע לא תקין של המערכת או מזהה עובד אי חיובי, נזרקות שגיאות.
2. בדיקה עם קיימות חברות במערכת - $O(1)$,בדיקה אם העובד קיים במאגר - $O(logn)$, במקרה ולא נזרקת שגיאה.
3. ניצור אובייקט מטיפוס Employee שיחזיק בתוכו את המידע המלא של העובד. – $O(logn)$.
4. נמחק מעץ של עובדי המערכת - $O(logn)$.
5. נמחק מעץ המשכורות של המערכת - $O(logn)$.
6. נמצא את החברה בה עובד העובד בעץ settledCompanies - $O(logk)$.
7. נסיר את העובד משני עצי חברה , נסמן m כמספר העובדים בחברה - $O(logm) + O(logm)$.
8. נבדוק אם החברה שבה עבד העובד ריקה מעובדים כעת - $O(logk)$, במידה ולא , נחזיר SUCCES, במידה וכן נסיר את החברה מעץ החברות הפעילות – SettledCompanies - $O(logk)$.
9. יצירת אובייקט מטיפוס Employee - $O(1)$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן :

$$O(logn) + O(logn) + O(logn) + O(logn) + O(logk) + O(logm) + O(logm) + O(logn) + O(logk) \leq 9logn = \boldsymbol{O(logn)}$$

סיבוכיות מקום: חיפוש ומחיקה בעצים הן פונקציות רקורסיביות ולכן $O(logn)$.

5) StatusType RemoveCompany(void *DS, int CompanyID) -

החברה פושטת רגל ויש למחוק אותה מהמערכת.

1. אם נשלח מצביע לא תקין של המערכת או מזהה חברה אי חיובי, נזרקות שגיאות.
2. נבדוק אם קיימות חברות במערכת - $O(1)$, במידה ולא , תיזרק שגיאה.
3. נבצע חיפוש בעץ Companies על מנת לבדוק אם החברה קיימת - $O(logk)$.
במידה ולא , תיזרק שגיאה.
4. נבצע חיפוש בעץ SettledCompanies על מנת לבדוק אם לחברה יש עובדים - $O(logk)$.
במידה ונמצא את החברה בעץ הזה סימן שיש עובדים בחברה, תיזרק שגיאה.
5. נבצע הסרה של החברה מהעץ companies - $O(logk)$.

סיבוכיות זמן : $O(1) + O(logk) + O(logk) + O(logk) = 1 + 3logk = \boldsymbol{O(logk)}$

סיבוכיות מקום: חיפוש ומחיקה בעצים הן פונקציות רקורסיביות ולכן $O(logk)$.

6) StatusType GetCompanyInfo(void *DS, int CompanyID, int *Value, int *NumEmployees) –
מחזירה את שווי החברה ומספר העובדים בה.

1. אם נשלח מצביעים לא תקינים של המערכת/שווי חברה/כמות עובדים או מזהה חברה אי חיובי, נזרקות שגיאות.
2. נבדוק אם קיימות במערכת חברות - $O(1)$, אם לא , תיזרק שגיאה.
3. נבדוק אם החברה קיימת ע"י חיפושה בעץ companies. - $O(logk)$, אחרת תיזרק שגיאה.
4. אם מצאנו נעדכן את המצביע לשווי החברה להיות data של החברה בעץ companies $O(logk)$.
5. נבצע חיפוש בעץ SettledCompanies $O(logk)$.
6. אם נמצא את בחיפוש נעדכן את המצביע של NumEmployees לכמות העובדים בחברה ואם לא שיצביע לאפס.$O(1)$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן : $O(1) + O(logk) + O(logk) + O(logk) + O(1) = 3logk + 2 = \boldsymbol{O(logk)}$.

סיבוכיות מקום: חיפוש בעצים הן פונקציות רקורסיביות ולכן $O(logk)$.

7) StatusType GetEmployeeInfo(void *DS, int EmployeeID, int *EmployerID, int *Salary, int *Grade) -
פונקציה המחזירה את מעסיקו, שכרו והדרגה הנוכחית של העובד.

1. אם נשלח מצביעים לא תקינים של המערכת/שכר/דרגה או מזהה עובד אי חיובי, נזרקות שגיאות.
2. נבדוק ונשמור מצביע, אם קיים עובד במערכת על ידי חיפוש בעץ idSortedEmp - $O(logn)$, במקרה ולא תיזרק שגיאה.
3. השמה עבור כל המצביעים. $O(1)$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן : $O(logn) + O(1) = \boldsymbol{O(logn)}$.

סיבוכיות מקום: חיפוש בעצים היא פונקציה רקורסיבית ולכן $O(logn)$.

8) StatusType IncreaseCompanyValue(void *DS, int CompanyID, int ValueIncrease) –
פונקציה תשנה את ערך החברה .

1. אם נשלח מצביע לא תקין של המערכת או מזהה חברה/תוספת לשווי אי חיוביים, נזרקות שגיאות.
2. נחפש בעץ companies את החברה וניצור מצביע אל האובייקט, במקרה ולא נמצא את החברה תיזרק שגיאה- $O(logk)$.
3. נבדוק את הdata של החברה ,כאשר באותו עץ הdata הוא שווי החברה, ונעדכן אותו בערך החדש- $O(1)$.
4. לאחר מכן נבדוק אם החברה קיימת בעץ של settledCompanies , אם לא נסיים ונשלח SUCCESS, אחרת, נעדכן את שווי החדש של החברה . $O(logk)$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן : $O(logk) + O(1) + O(logk) + O(1) = \boldsymbol{2logk + 2} = \boldsymbol{O(logk)}$.

סיבוכיות מקום: חיפוש בעצים היא פונקציה רקורסיבית ולכן $\boldsymbol{O(logk)}$.

9) StatusType PromoteEmployee(void *DS, int EmployeeID, int SalaryIncrease, int BumpGrade)-
העובד מקבל קידום ועל הפונקציה לעדכן זאת במערכת.

1. אם נשלח מצביע לא תקין של המערכת או מזהה עובד/תוספת לשכר אי חיוביים, נזרקות שגיאות.
2. נשמור מצביע ונבדוק אם העובד נמצא במערכת - $O(logn)$. ואם לא תיזרק שגיאה.
3. לאחר מכן נעדכן את השכר החדש של העובד ואת דרגתו אם BumpGrade חיובי, אחרת , הדרגה לא תשתנה. – O(1).
4. נחפש את העובד בעץ המשכורות של המערכת -$O(logn)$.
5. נמחק את האובייקט הקיים- . $O(logn)$.
6. נקצה אובייקט מעודכן ונכניס אותו לעץ המשכורות של המערכת- . $O(logn)$.
7. ניצור מצביע לחברה שבה עובד העובד על ידי חיפוש בsettledCompanies - $O(logk)$.
8. עבור עץ המשכורות של החברה נמחק את האובייקט הישן- . $O(logm)$ כאשר m מייצג את מספר העובדים בחברה.
9. נמחק את האובייקט הישן- . $O(logm)$ כאשר m מייצג את מספר העובדים בחברה.
10. נקצה אובייקט מעודכן ונכניס אותו לעץ המשכורות של החברה- . $O(logm)$ כאשר m מייצג את מספר העובדים בחברה.
11. נעדכן את פרטי העובד בעץ בעובדים של החברה הממוין לפי מספרים מזהים- $O(logm)$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן : $O(logn) + O(1) + O(logn) + O(logn) + O(logn) + O(logk) + O(logm) + O(logm) + O(logm) + O(logm) = \boldsymbol{4logn + logk + 4logm + 1} = \boldsymbol{O(logn)}$.

סיבוכיות מקום: חיפוש בעצים היא פונקציה רקורסיבית ומכיוון שמדובר על חיפוש בעצי עובדים $\boldsymbol{O(logn)}$

10) StatusType HireEmployee(void *DS, int EmployeeID, int NewCompanyID)
חברה מגייסת עובד מחברה אחרת, הפונקציה מבצעת את השינויים המתאימים במערכת.

1. אם נשלח מצביע לא תקין של המערכת או מזהה עובד/חברה אי חיוביים, נזרקות שגיאות.
2. נבצע 2 בדיקות : אם לא קיימים חברות במערכת ואם לא קיימים עובדים במערכת יזרקו שגיאות -$O(1)$.
3. ניצור מצביע לEmployee על ידי חיפושו בעץ העובדים של המערכת - $O(logn)$.
4. ניצור מצביע לחברה החדשה אליה עובר העובד O(logk) , במקרה והמצביעים יצביעו על nullptr תיזרק שגיאה.
5. מקרה נוסף שתיזרק שגיאה : החברה אליה צריך לעבור העובד היא החברה שבה הוא עובד, הבדיקה O(1).
6. נשמור את הפרטים הרלוונטים O(1).
7. מחיקת העובד מהמערכת – $O(logn)$
8. יצירת עובד חדש עם פרטים מעודכנים - $O(logn + logk)$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן :

$$O(logn) + O(logk) + O(1) + O(logn) + O(logn + logk) = \boldsymbol{2logn + logk + 1 + (logn + logk)} = \boldsymbol{O(logn + logk)}$$

סיבוכיות מקום: שימוש ב addEmployee $O(logn + logk)$.

11) StatusType AcquireCompany(void *DS, int AcquirerID, int TargetID, double Factor) -
חברה אחת רוכשת את החברה השניה, הפונקציה תבצע את השינויים המתאימים ותעדכן את המערכת .

1. במקרה ונשלח מצביע לא תקין של המערכת / מספרים מזהים של החברות אי חיוביים/factor קטן מ-1 , נזרקות שגיאות.
2. בדיקה אם קיימות חברות במערכת - $O(1)$.
3. ניצור מצביעים ונחפש את כל אחת מהחברות בעץ companies - $O(logk)$ לכל חיפוש, אם לא נמצא לפחות חברה אחת תיזרק שגיאה.
4. נבצע בדיקה ששווי החברה הרוכשת גדול פי 10 לפחות משווי החברה הנרכשת, אם המצב אינו כך , תיזרק שגיאה. $O(1)$.
5. נעדכן את השווי החברה הרוכשת $O(1)$.
6. באותו אופן , ניצור מצביעים לחברות ע"י הסריקות בעץ settledCompanies - $O(logk)$.
7. כעת ישנם כמה מצבים אפשרים:
   1. לחברה הנרכשת אין עובדים, במקרה זה נעדכן את שווי החברה הרוכשת בלבד.- O(1) .
   2. לחברה הרוכשת אין עובדים, במקרה זה , נקצה אובייקט מטיפוס Company, ובנוסף נקצה עץ עובדי חברה ועץ משכורות בחברה - $O(1)$, ונשתמש במתודה mergeCompanies -$O(n_{target})$.
   3. במקרה ולשני החברות יש עובדים , נבצע mergeCompanies שמשתמשת בmergeTree בעלת סיבוכיות ריצה $O(n_{target} + n_{acquirer})$.
8. נמחק את החברה שנרכשה בעץ settledCompanies במקרה והיו לה עובדים- $O(n_{target})$.
9. נמחק את החברה שנרכשה מעץ companies - $O(n_{target})$.
10. נשתמש בפונקציה עזר adjustCompId שהיא פונקציה רקורסיבית העוברת על כל העץ ומעדכן את הפוינטר לחברה בה עובדים להצביע למזהה של החברה שרכשה- $O(n_{target} + n_{acquirer})$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן :

$$O(1) + O(logk) + O(1) + O(1) + O(logk) + O(n_{target} + n_{acquirer}) + O(n_{target}) + O(n_{target}) + O(n_{target} + n_{acquirer}) = \boldsymbol{2logk + 3 + 4n_{target} + 2n_{acquirer}} = \boldsymbol{O(logk + n_{target} + n_{acquirer})}$$

סיבוכיות מקום: במקרה הגרוע ביותר קיימים בזמן נתון 2 מערכים בגודל $\boldsymbol{n_{target} + n_{acquirer}}$ ומשתמשים בArrayToTree פונקציה רקורסיבית ולכן: $O(\boldsymbol{n_{target} + n_{acquirer}} + \log(\boldsymbol{n_{target} + n_{acquirer}}))$.

12) StatusType GetHighestEarner(void *DS, int CompanyID, int *EmployeeID) –

1. במקרה ונשלח מצביע לא תקין של המערכת / מצביע למספר מזהה של העובד או מספר מזהה של החברה אי חיובי נזרקות שגיאות. בנוסף אם לא קיימות חברות או עובדים במערכת אז יזרקו שגיאות.- $O(1)$.
2. אם מזהה חברה חיובי , ניצור מצביע ונחפש את החברה בעץ companies- $\boldsymbol{O(logk)}$, במקרה ואינה שם תיזרק שגיא
   1. באותו אופן, ניצור מצביע ונחפש את החברה בעץ settledCompanies- $\boldsymbol{O(logk)}$, אם נמצא אותה בעץ המשמעות היא שיש לחברה עובדים, במקרה ולא נמצא תיזרק שגיאה.
   2. נשתמש בפוויינטר שיש לכל עץ אשר מצביע לאובייקט המקסימלי בעץ (לפי איך שהעץ ממוין) – **O(1)** .
3. אם נזהה החברה שלילי :
   1. במערכת ניגש לעץ משכורות כלל העובדים , ונשתמש בפוויינטר על מנת לשלוף את העובד בעל המשכורת הגבוהה ביותר – **O(1)**, אם עץ המשכורות של המערכת ריק , תיזרק שגיאה.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן :

$$O(1) + O(logk) + O(logk) + O(1) = O(logk) \ , if\ CompanyID > 0$$
$$O(1), if\ companyID < 0$$

סיבוכיות מקום: חיפוש בעצים היא פונקציה רקורסיבית , אנחנו מבצעים חיפוש בעצי חברות, ולכן $O(logk)$.

13) StatusType GetAllEmployeesBySalary(void *DS, int CompanyID, int **Employees, int *NumOfEmployees) –

1. אם מזהה חברה חיובי , ניצור מצביע ונחפש את החברה בעץ companies- $\boldsymbol{O(logk)}$, במקרה ואינה שם תיזרק שגיא
   1. באותו אופן, ניצור מצביע ונחפש את החברה בעץ settledCompanies- $\boldsymbol{O(logk)}$, אם נמצא אותה בעץ המשמעות היא שיש לחברה עובדים, במקרה ולא נמצא תיזרק שגיאה.
   2. נקצה מערך בגודל מספר האובייקטים בעץ המשכורות של החברה – O(1) .
   3. נבצע InOrder , כאשר הפלט יכנס בעצם למערך – $O(\boldsymbol{n_{CompanyID}})$.
2. במקרה ומזהה החברה שלילי , נבצע תהליך דומה רק עבור עץ משכורות של כלל העובדים במערכת , ההבדל הוא שגודל המערך יהיה n , וסיבוכיות הזמן של inOrder תהיה O(n) .
3. כעת , נקצה מערך בגודל len , len הוגדר בתחילת המתודה ומחזיק בתוכו את ערך של מספר האובייקטים בעץ.
4. נבצע היפוך למערך , מכיוון שעלינו להחזירו בצורת Reverse inOrder - $O(n)/O(\boldsymbol{n_{CompanyID}})$ (תלוי מקרה)
5. Employees יצביע למערך הסופי. –O(1) .

חשוב להדגיש כי לאורך כל המתודה, בעת הקצאת מערכים/ אובייקטים , אם והקצאות נכשלו תיזרק שגיאה, אם כל התהליך צלח נחזיר SUCCESS.

<u>סיבוכיות זמן :</u> במקרה הגרוע ביותר עבור כל אחד מהמקרים הבאים:

$$O(logk) + O(logk) + O(1) + O(\boldsymbol{n_{CompanyID}}) + O(\boldsymbol{n_{CompanyID}}) + O(1)$$
$$= O(\log k + \boldsymbol{n_{CompanyID}}) \ , if\ CompanyID > 0$$
$$O(1) + O(\boldsymbol{n}) + O(\boldsymbol{n}) + O(1) = O(n), if\ companyID < 0$$

<u>סיבוכיות מקום:</u> במקרה הגרוע ביותר קיימים 2 מערכים כאשר כל אחד מהם בגודל n ומתבצעת פונקציה רקורסיבית inOrder , ולכן $O(n) + O(n) + O(logn) = O(n + logn), if\ CompanyID < 0$.

$$O(n_{CompanyID} + logn_{CompanyID}), if\ CompanyID > 0$$

14) StatusType GetHighestEarnerInEachCompany(void *DS, int NumOfCompanies, int **Employees) -

1. נבדוק מה גודל העץ settledCompanies , ואם במקרה הגודל הוא אפס , תיזרק שגיאה. O(1).
2. נקצה מערך integers בגודל NumOfCompanies שיכיל את מזהי החברות , במקרה ונכשלה פעולת ההקצאה תיזרק שגיאה.$O(1)$
3. נקצה מערך מצביעים לאובייקטים מטיפוס Company בגודל NumOfCompanies שיכיל את החברות עצמן, במקרה ונכשלה פעולת ההקצאה תיזרק שגיאה.$O(1)$
4. נבצע InOrder לעץ settledCompanies , InOrder בנוי לרוץ רקורסיבית כגודל המערך ששלחו אליו, במקרה זה NumOfCompanies פעמים.$O(NumOfCompanies)$
5. נשחרר את מערך הintegers.- $O(NumOfCompanies)$
6. נקצה מחדש את המערך integers בגודל NumOfCompanies , אליו נכניס את הmax_node של עץ המשכורות של כל חברה. $O(NumOfCompanies)$.
7. נשחרר את מערך ה company – $O(NumOfCompanies)$
8. Employees יצביע אל המערך integers.

חשוב להדגיש כי לאורך כל המתודה, בעת הקצאת מערכים/ אובייקטים , אם והקצאות נכשלו תיזרק שגיאה, אם כל התהליך צלח נחזיר SUCCESS.

<u>סיבוכיות זמן :</u> במקרה הגרוע ביותר עבור כל אחד מהמקרים הבאים:

$$O(1) + O(1) + O(1)O(NumOfCompanies) + O(NumOfCompanies) + O(NumOfCompanies)$$
$$+ O(NumOfCompanies) + O(NumOfCompanies) = 3 + 5NumOfCompanies$$
$$= \boldsymbol{O(NumOfCompanies)} \leq O(logk + NumOfCompanies)$$

<u>סיבוכיות מקום:</u> במקרה הגרוע ביותר קיימים 2 מערכים כאשר כל אחד מהם בגודל $NumOfCompanies$ ומתבצעת פונקציה רקורסיבית inOrder מספר של $\log(NumOfCompanies)$ פעמים, ולכן:

$$O(NumOfCompanies) + O(NumOfCompanies) + O(\log(NumOfCompanies))$$
$$= O(NumOfCompanies + \log(NumOfCompanies))$$

15) StatusType GetNumEmployeesMatching(void *DS, int CompanyID, int MinEmployeeID, int – MaxEmployeeId, int MinSalary, int MinGrade, int *TotalNumOfEmployees, int *NumOfEmployees)

1. בדיקה אם אין חברות / עובדים במערכת, אם אין יזרקו שגיאות - $O(1)$.
2. יהיה לנו מונה של מספר עובדים שיאותחל ל-0, והוא יקרא numEmployees - $O(1)$.
3. אם ComanyID>0 אז נבצע את הפעולות הבאות:
   1. ניצור מצביע ונחפש את החברה בעץ companies של המערכת, אם לא נמצא את החברה תיזרק שגיאה. $O(logk)$
   2. ניצור מצביע ונחפש את החברה בעץ settledCompanies של המערכת , אם לא נמצא את החברה תיזרק שגיאה.$O(logk)$.
   3. כעת נבדוק כמה מתוך העובדים בחברה נמצאים בטווח של מזהי העובדים הרלוונטים, פעולה זאת מתבצעת ע"י המתודה countCondNodesAss המפורטת בפונקציות עזר נוספות שמימשנו עבור מחלקת העץ, המתודה מחזירה את מספר העובדים שעומדים בתנאים – TotalNumOfEmployees - $O(\text{TotalNumOfEmployees})$.
   4. אם מספר העובדים שווה ל-0 , אז $NumOfEmployees$ יצביע ל-0, המתודה תסתיים ותחזיר SUCCESS.

   אחרת, נקצה מערך מצביעים בגודל TotalNumOfEmployees,של מצביעים מטיפוס Employee. ונשתמש במתודה specialCondNodeAss שתמלא את מערך המצביעים שעומדים בתנאי הטווח - $O(\text{TotalNumOfEmployees})$.
4. אם CompanyID<0 אז נבצע את הפעולות הבאות:
   1. אל מצביע TotalNumOfEmployees נבצע השמה בעזרת המתודה countCondNodesAss שתתבצע על עץ העובדים של כלל המערכת הממוין על פי EmpoleeID. $O(\text{TotalNumOfEmployees})$.
   2. אם TotalNumOfEmployees שווה ל-0 , נבצע השמה למצביע NumOfEmployees שיהיה שווה לאפס ואז תסתיים המתודה ויחזור SUCCESS.

   אחרת, נקצה מערך מצביעים בגודל TotalNumOfEmployees אשר מצביעים אל טיפוסי Employee. ונשתמש במתודה specialCondNodeAss שתמלא את מערך המצביעים שלEmployees שעומדים בתנאי הטווח. $O(\text{TotalNumOfEmployees})$.
5. כעת עבור 2 המקרים נבצע את הפעולות הבאות:
   1. נרוץ על מערך הEmployees ונעדכן את המונה numEmployees במקרה וה Employee עומד בדרישה של שכר וגם של הדרגה ← numEmployee++. -$O(\text{TotalNumOfEmployees})$
6. *NumOfEmployees=numEmployess , ונשחרר את מערך המצביעים - $O(1)$.

חשוב להדגיש כי לאורך כל המתודה, בעת הקצאת מערכים/ אובייקטים , אם והקצאות נכשלו תיזרק שגיאה, אם כל התהליך צלח נחזיר SUCCESS.

<u>סיבוכיות זמן :</u> במקרה הגרוע ביותר עבור כל אחד מהמקרים הבאים:

$$O(logk) + O(logk) + O(\text{TotalNumOfEmployees}) + O(\text{TotalNumOfEmployees}) + O(1)$$
$$= O\left(logk + logn_{CompanyID} + TotalNumOfEmployees\right) \ ,if\ CompanyID > 0$$

חשוב להסביר כי לפי ההסבר של הפונקציות *CountNodes* ו- *specialCountNodes* במקרה שבו בטווח מזהי העובדים נמצאים כל העובדים אזי $TotalNumOfEmployees = n_{CompanyID}$ ולכן מתקיים השוויון למעלה.

$$O(\text{TotalNumOfEmployees}) + O(\text{TotalNumOfEmployees}) + O(\text{TotalNumOfEmployees}) + O(1)$$
$$= \boldsymbol{O(\text{TotalNumOfEmployees})} \leq O(logn + TotalNumOfEmployees) \quad ,if\ companyI < 0$$

<u>סיבוכיות מקום:</u> במקרה הגרוע ביותר קיימת הקצאה של מערך מצביעים בגודל TotalNumOfEmployees כאשר במקרה הגרוע ביותר אלה כלל העובדים במערכת $O(n)$, מתבצעות פונקציות רקורסיביות *CountNodes* ו- *specialCountNodes* , ולכן:

$$O(\text{TotalNumOfEmployees}) + O(\log(\text{TotalNumOfEmployees}))$$
$$= \boldsymbol{O(log(\text{TotalNumOfEmployees}) + \text{TotalNumOfEmployees})}$$

16) void Quit(void **DS) –

משחררת את המבנה ואת כל הזיכרון שהוקצה , כל מחיקה של עץ מתבצעת ע"י המתודה deleteAllNodes ואז קריאה לדיסטרקטור, סיבוכיות הזמן של deleteAllNodes היא $O(t)$, כאשר t הוא מספר האובייקטים בעץ.

1. מחיקת עץ companies $O(k+n)$.
2. מחיקת עץ settledCompanies $O(k+n)$.
3. מחיקת עץ salarySortedEmp - $O(n)$.
4. מחיקת עץ idSortedEmp - $O(n)$.
5. שחרור המערכת - $O(1)$.
6. *DS=nullptr

<u>סיבוכיות זמן :</u> $O(n+k)$.

<u>סיבוכיות מקום:</u> deleteAllNodes היא פונקציה רקורסיבית וקוראים לה $O(logn + logk)$.

<u>סיבוכיות מקום של התוכנית:</u>

הפונקציה עם סיבוכיות מקום הגרועה ביותר מכלל הפונקציות היא בעלת סיבוכיות מקום: $\boldsymbol{O(log(\text{TotalNumOfEmployees}) + \text{TotalNumOfEmployees})}$

כאשר במקרה הגרוע ביותר כל העובדי המערכת עובדים באותה חברה, ***TotalNumOfEmployees=n*** אז: $\boldsymbol{O(logn + \text{n}) \leq O(2n) = O(n)}$.

סה"כ בכל רגע נתון המערכת מחזיקה: ***n*** עובדים ו-***k*** חברות, 4 עצים אשר מחזיקים: $\boldsymbol{k + n + k + n + n = 3n + 2k = O(n+k)}$ אובייקטים ולכן עומד בדרישות סיבוכיות המקום.