שאלה 1:
א)
מנגנון RTTI שזה קיצור ל Run-time type information הוא הוספת מפתח לכל ערך בזיכרון שיציין את הטיפוס של הערך, המפתח יכול להיות רפרנס ל "type descriptor" או יכול להכיל את כל המידע על טיפוס, בדרך כלל אחיד לכל הערכים בזיכרון מאותו טיפוס והמפתח הזה יפוענח בזמן ריצת התוכנית בשפות דינמיות וידוע לקומפיילר בשפות סטטיות.

ב)
כשמשתמשים במנגנון RTTI אנחנו נשלם על זה מחיר במקום (לכל טיפוס חייב להיות vtable) ומשלמים על זה זמן.
בשפת C קיים עקרון "no hidden cost" שהוא שהמתכנת לא משלם על דברים שאינם חשופים לו, RTTI הוא נמצא בשפה ולא חשוף למתכנת ואין לו שליטה עליו אבל הוא משלם עליו, וזה סותר את העיקרון הנזכר למעלה.

ג)
ב garbage collection אנחנו לא רק צריכים למצוא את הערכים שהקצינו אלא יתר על זאת אנחנו חייבים הטיפוסים של הערכים הקשורים אליהם, בזיכרון יש לנו בתים וביטים אבל אנחנו לא יודעים מה הם מייצגים מנגנון RTTI יעזור לנו להתגבר על דבר זה ואז אנחנו נדע מה הערכים המקושרים לערכים שהקצינו.

ד)
למשל ב deep cloning, לכל ערך אנחנו ניגש לכל ערך ישיג ממנו, בלי ה RTTI אנחנו לא נדע את הטיפוס של ערכים אלה ואז לא נדע איך לגשת להם מתוך הבלוק המכיל בתים.

שאלה 2:
א)

difference	similarity
גודל המערך הסטטי ידוע בזמן קומפילציה לעומת המערך הדינמי שיוחלט על גודלו בזמן ריצה	הגודל של המערך לא משתנה אחרי האתחול
המערך הסטטי מוקצה על ה data segment המערך הדינמי מוקצה על ה heap segment	

ב)

difference	similarity
האינדקסים ב stack based array הם יוחלטו בזמן יצור המערך אבל ב associative array הם יכולים להיות כל דבר ואז הסט שלהם לא חסום	
הגודל של stack based array הוא לא ישתנה במהלך ריצת התוכנית אבל associative array הגודל שלה יגדל ויקטן לפי מספר האיברים בה (כי היא בדרך כלל ממושתת כטבלת ערבול.	

כיוון ש associative array דורשת מבנה נתונים מתוחכם ולא פשוט היא לא יעילה כמו stack based array	

ג)
אלגוריתם Mark-compact Algorithm

1. נסמן את העצמים שאנחנו יכולים להגיע אליהם.
2. נקצה מחדש את כל העצמים לתחילת ה heap.
3. נעדכן את המצביעים לעצמים שהקצינו מחדש.

נראה את היתרונות והחסרונות ל Mark-compact Algorithm עם mark and sweep

cons	pros
Throughput איטי יותר, בדרך כלל נצטרך כמה פעמים לעבור על הערימה	זמן ריצת ה GC הוא $O(live\ objects)$ לעומת $O(heap\ size)$ב mark and sweep
	הפרגמנטציה קטנה
	מימוש קל

ראה את היתרונות והחסרונות ל stop and copy ביחס לניהול זיכרון ידני

cons	pros
	חיים קלים למתכנת כי הוא לא ישחרר זיכרון בעצמו
מאבדים יעילות של התוכנית בזמן ריצת GC הדבר הזה שיכולים לחסוך בו ע"י תכנות יותר טוב	לא יהיה heap corruption או memory leaks או dangling reference

שאלה 3)

א)
שגיאות מסוג type error (מתרחשות בשפות שהן dynamic typed, כלומר שאין בהם type checking) הם שגיאות שבהם בוצעה פעולה על ערך כלשהו שאינה תואמת את הטיפוס שלו. למשל עבור שפות :
"mohammed" + 5

שגיאות מסוג pseudo type error (מתרחשות גם בשפות dynamic typed וגם בstatic typed) הם שגיאות שבהם הפעולה המבוצעת על הערך אינה תואמת את הערך (למשל חלוקה ב0 או חריגה מגבולות מערך וכו').
כאמור ההבדל בין type error לpseudo type error הוא שtype error יזרק עבור כל ערך מהטיפוס הנתון, וpseudo type error תזרק רק עבור חלק מהערכים מהטיפוס הנתון .

ב)

שיטות השערוך שלמדנו)

- Eager - כאשר כל הפרמטרים של הפונקציה משוערכים טרם תחילת ריצתה (הסדר תלוי בשפה, אבל הפונקציה מופעלת ע"י ה Caller)
- short circuit בשערוך של הביטויים לוגים שמכילים OR או AND אבהן הארגומנט השני יחושב רק אם הראשון לא מספיק לקביעת ערך הביטוי הכולל .
- Normal הארגומנט לפנוקציה ישוערך בכל פעם שיעשה בהם שימוש .
- Lazy ארגומנט לפונקציה ישוערך בפעם הראשונה שבה ישעה בו שימוש בפונקציה ורק בפעם הראשונה.

דוגמה:

```
unsigned int myFun(unsigned int x){

unsigned int m = x;

While(x != 0){x--;}
Return m;
}
```

ג)

שפה אורתוגונלית היא שפה שניתן להפעיל כל בנאי על כל טיפוס .

נוכל להסיק שבנאי אורתוגונלי הוא בנאי שיכול לפעול על כל טיפוס .