הדרישות ההכרחיות שעל הטיפוס T לקיים :

I. אופרטור > : על מנת לשמור על מיון הרשימה, קיים הכרח לקיום אופרטור השוואה בין שני משתני T על מנת להכריע באיזה סדר יופיעו.
II. קיום הורס: על מנת להרוס אותו כאשר אין צורך בו יותר.
III. קיום בנאי העתקה :על מנת ליצור עותק משלנו לכל משתנה T שנקבל מהמשתמש, כדי לנתק את התלות מניהול הזיכרון של המשתמש.

2. במידה והיינו רוצים לממש non-const iterator עבור Sorted List, הבעיה היא שהמימוש נשען על כך שבכל רגע נתון הרשימה ממוינת, כלומר כל איבר נמצא במקום המתאים לפי אופרטורי ההשוואה > של המחלקה T .

כאשר אנו נותנים למשתמש אפשרות לשנות באמצעות האיטרטור את הערך עליו האיטרטור מצביע, ייתכן כי הרשימה לא תהיה ממוינת יותר. המבנה הפנימי של הרשימה שממומשת באמצעות צמתים (NODE) יישאר זהה גם לאחר השינוי של הערך (כל צומת תצביע על אותה צומת לפני שינוי הערך) אך ייתכן כי הערך לא יהיה במקום המתאים עבורו ברשימה.

לכן אם היינו מרשים איטרטור שהוא אינו oconst היינו צריכים לשנות את המימוש כך שנסדר את הרשימה בכל פעם שהיה שימוש באופרטור *.

3. ניתן להעביר את הפרדיקט לפונקציה A (filter לדוגמה) כמצביע לפונקציה בוליאנית כפרמטר של A, או ע"י שימוש בתבנית והעברת Function Object . (כלומר העמסת האופרטור () של מחלקה בשם מסוים והעברת עצם מאותה מחלקה לפונקציה A כפרמטר) .

בשיטה הראשונה, ניצור פונקציה שמקבלת כמות וסוג פרמטרים מסוימים, ונצטרך להעביר לה את כולם כאשר נשתמש בה. בשיטה השנייה, בגלל שייתכנו מספר רב של בנאים ושל פעולות של המחלקה, נוכל להעביר לאותה עצם כמות שונה וסוגים שונים של פרמטרים בהתאם לבנאים שיצרנו למחלקה, ולכן נוכל להפעיל מגוון רחב יותר של פעולות.

עם זאת, לאחר שבחרנו באחת השיטות ומימשנו את A, במידה ונרצה להחליף לשיטה השנייה אין צורך לשנות את המימוש של A :

1. במידה ונרצה להחליף למצביע לפונקציה נוכל ליצור פונקציה בשם זהה שמקבלת אותם פרמטרים כמו הFunction Object הקיים ועושה אותה פעולה של האופרטור () של המחלקה.
2. במידה ונרצה להחליף לFunction Object נממש מחלקה עם בנאי המקבל פרמטרים זהים לפונקציה הקיימת ונעמיס את האופרטור () בצורה דומה לפעולת הפונקציה.